1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1/29/92	Final Report. 7/1/91 - 12/31/91

4. TITLE AND SUBTITLE
Scalable Parallel Algorithms for Multidimensional Digital Signal Processing

5. FUNDING NUMBERS
DAAL03-91-C-0038

6. AUTHOR(S)
Dr. Martin Rofheart
Dr. John Granata

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Iterations, Inc.
450 Seventh Avenue, Suite 4302
New York, NY 10123

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 29087.1-MA-SBI

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A new algorithm for the multiprocessor computation of the multidimensional discrete Fourier transform is presented. The algorithm eliminates the need for interprocessor communication and is highly scalable. The cost of eliminating interprocessor communication by this method is that one addition must be performed at each processing element for every input loaded into the machine.

The algorithm is based on a new multidimensional discrete Fourier transform algorithm that computes a d-dimensional discrete Fourier transform by a set of independent k-dimensional discrete Fourier transforms (k<d); it is a reduction algorithm in the sense that it has lowered the dimension of the Fourier transforms that are computed. The k-dimensional discrete Fourier transforms are performed on data derived from the input using only additions, and produce k-dimensional hyperplanes of the output array.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Multidimensional Discrete Fourier Transform Algorithms Parallel Algorithms	58
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Final Report: Scalable Parallel Algorithms for Multidimensional Digital Signal Processing

Iterations, Inc.
450 Seventh Avenue, 43rd floor
New York City, NY 10123
(212) 695 3709

December 31, 1991

Abstract - A new algorithm for the multiprocessor computation of the multidimensional discrete Fourier transform is presented. The algorithm eliminates the need for interprocessor communication and is highly scalable. The cost of eliminating interprocessor communication by this method is that one addition must be performed at each processing element for every input loaded into the machine.

The algorithm is based on a new multidimensional discrete Fourier transform algorithm that computes a $d-$dimensional discrete Fourier transform by a set of independent $k-$dimensional discrete Fourier transforms ($k < d$); it is a reduction algorithm in the sense that it has lowered the dimension of the Fourier transforms that are computed. The $k-$dimensional discrete Fourier transforms are performed on data derived from the input using only additions, and produce $k-$dimensional hyperplanes of the output array.

The mapping of the algorithm onto architectures with broadcast and report capabilities is given. Expressions are obtained for estimating the speed on these machines as a function of the size of the $d-$dimensional DFT, the bandwidth C of the communications channel, the time A for an addition, the time $T(FFT)$ for a single processing element to perform a $k-$dimensional DFT ($k < d$), and the degree of parallelism of the machine. For single I/O channel machines that are capable of exploiting the full degree of parallelism of the algorithm, execution times as low as the time required to compute a single $k-$dimensional DFT plus the I/O time for data upload and download are attainable.

Supported by
Strategic Defense Initiative/Innovative Science and Technology
Managed by US Army Research Office Under Contract No. DAALO3-91-C-0038

Contents

1 *Program Overview* **1**
- 1.1 *Applications* . . . 2
- 1.2 *Background* . . . 3
- 1.3 *Reduced Transform Algorithms* . . . 6
- 1.4 *Report Organization* . . . 8

2 *An Algorithm for the Computation of the MD DFT on Hyperplanes* **9**
- 2.1 *Definitions* . . . 10
- 2.2 *Prime Case* . . . 12
- 2.3 *Power of Prime Case* . . . 16
- 2.4 *Appendix* . . . 22

3 *An Algorithm for the Multiprocessor Computation of MD DFT* **28**
- 3.1 *Introduction* . . . 28
- 3.2 *Algorithm Definition* . . . 29

4 *A Hybrid Algorithm for the Multiprocessor Computation of MD DFT* **42**
- 4.1 *Derivation* . . . 43
- 4.2 *Conclusion* . . . 50

5 *Program Results* **51**

6 *References* **53**

Chapter 1

Program Overview

This program presents a new algorithm for the multiprocessor computation of the multidimensional discrete Fourier transform (MD DFT). The main features of the algorithm are: (1) that it requires no interprocessor communication, and (2) that it is highly scalable. The cost of eliminating interprocessor communication by this method is that one addition must be performed at each processing element for each input broadcast to the machine.

The motivation for this research program has been the development of algorithmic methods that can exploit the power of VLSI based multiprocessors. These machines have large computational capabilities but limited communication bandwidth. They strongly favor algorithms that minimize interprocessor communications. This principle of locality [32] dominates algorithm design at all levels.

Multidimensional Cooley-Tukey algorithms, and their variants, require interprocessor communication because they partition the data set at every stage of the computation. At a minimum this necessitates an interprocessor communication requirement where every processing element must exchange data with every other processing element to complete the calculation.

Most parallel algorithms for MD DFT computation attempt to minimize the interprocessor communications requirement inherent in Cooley-Tukey methods. This research program has taken a new approach. The algorithm presented does not apply a partition to the input data. Rather, it performs a reduction operation in parallel on the data. This reduction requires each processing element to perform one addition for each word loaded into the machine. The computation is completed by performing lower-order/lower-dimension MD DFT on the data, derived by the additions, at each processing element. For a $d-$dimensional DFT each processor would be required to compute a $k-$dimensional DFT where k is one of $k = 0, 1, \ldots, d-1$ depending on the degree of parallelism of the implementation. The algorithm eliminates the requirement of the interconnection network that typically dominates parallel computation of the MD DFT at a cost of one addition at each processing element for each word loaded into the machine.

The algorithm is based on a parallelization of the hyperplane algorithm for MD DFT computation presented in chapter 2. Two modes of parallelization are considered.

The first is a direct approach given in chapter 3. In this form the algorithm requires no interprocessor communication, however it is not scalable. A hybrid approach that marries multidimensional Cooley-Tukey type methods with the hyperplane algorithm is presented in chapter 4. The resulting hybrid algorithm requires no interprocessor communication and is highly scalable.

1.1 Applications

Computations that requires fast execution of MD DFT are well suited to parallel methods. Computational techniques based on conventional methods require extensive interprocessor communication which in turn require an interconnect network between processing elements. Interconnect networks are slow, large, expensive, and consume large amounts of power.

The hardware requirement for parallel MD DFT computation is drastically reduced by the elimination of the communication interconnection network. This allows low cost, low power, and small size machines to compute MD DFT at high speed.

In order to exploit the method developed in this program, the data flow of the target problem must match that of the algorithm. One such problem is the processing of synthetic aperture radar (SAR) data. SAR is an airborne radar imaging technique that generates high resolution terrain maps by processing radar returns [20]. A well known method of SAR image reconstruction interprets the SAR data as samples of the Fourier transform of the targeted terrain. The image is reconstructed by inverse transforming the sampled Fourier data. The realtime computation of the SAR image using this technique requires the realization of large scale 2D DFT and interpolations. Under current technology, parallel computation is required to achieve high resolution in realtime or near realtime. Realtime systems based on conventional digital signal processing techniques require extensive interprocessor communication to compute the 2D DFT. The required communication networks greatly increase the size, cost, and power consumption of those computing structures.

By the method developed in this research program, this complex structure could be replaced by a multiprocessor with only broadcast communications. The Fourier domain samples could be broadcast to a collection of processing elements where they are transformed into the spatial domain by the method described in this research. After the computation is completed, the image data would be distributed in the machine in the same manner as if a vector-radix MD DFT were employed.

Currently other applications are being vigorously explored to determine how suitable they are for multiprocessor computation by the method developed in this research program.

Implementations

For proof of concept a variant of this algorithm was implemented on the AT&T BT100 [2] multiprocessor. The implementation was benchmarked over ten times faster then a row-column 2D DFT of the same size on the machine. The cause of the dramatic difference in speed is that on the BT100 communication is much slower than computation.

1.2 Background

Algorithms that compute the multidimensional discrete Fourier transform (MD DFT) have been roughly categorized as row-column, multidimensional variants of Cooley-Tukey (MD CT), and reduced transform algorithms (RTA). Parallel implementations of the row-column and multidimensional Cooley-Tukey algorithms have been widely studied in the literature, a short list of these includes [19,18,37,13,24,6]. The feature of these algorithms most pertinent to this program is that they alternate stages of computation with stages of global data exchange.

The row-column algorithm evaluates the $N \times N$ 2D DFT by computing the 1D $N-$point DFT of the rows and columns of the 2D array. Parallel implementations of this algorithm require a global transpose operation between the row and column FFT stages. Specific implementations of this algorithm differ in the machine model and method used to perform the transpose. For a distributed machine model, the global transpose operation requires every processing element to exchange data with every other processing element. Interconnection networks that can transpose an $N \times N$ array distributed on N processors in $O(N)$ time include: the data manipulator[10], omega[21], Staran flip[5],generalized cube[30], and ADM[31]. The data manipulator requires $log_2 N$ stages of N switches, while the others require $log_2 N$ stages of $N/2$ switches. These systems are the most costly and complex elements of the multiprocessor system.

The Cooley-Tukey [9] algorithm was extended to multiple dimensions in [27,14,17]. A unified treatment of these is given in [22]. As in the one dimensional case, the additive properties of the indexing set are exploited to factor the MD DFT into alternating stages of lower order MD DFT, twiddle multiplication, and data permutation. This decomposition process can be applied recursively to produce algorithms whose dataflow is optimized for a specific architecture [18], and that have a reduced number of multiplies relative to the row-column method. However, all variants of the multidimensional Cooley-Tukey algorithm require alternating stages of DFT with stages of data exchange. Like the row-column method, extensive interprocessor communications is required.

Reduced transform algorithms (RTA) will be considered as alternative methods of computation. These algorithms apply number theory or polynomial theory to the computation of the multidimensional DFT. They are reductions in the sense that they compute a $d-$dimensional DFT with fewer 1D DFTs than the dN^{d-1} 1D DFTs required by the row-column method. These algorithms include the polynomial transform

algorithm [25,26], the multidimensional AFW algorithm [3], the weighted redundancy transform (WRT) [35,34], and the linear congruence algorithm [11]. Some RTA are shown to possess structures that do not intersperse computation stages with communication stages. The algorithms that satisfy this constraint can be implemented on distributed processors with no interprocessor communication. These algorithms have neither the time nor the hardware cost associated with the data exchange stages of row-column and multidimensional Cooley-Tukey algorithms.

The remainder of this chapter is organized as follows. First, the multidimensional DFT is defined, and the row-column and multidimensional Cooley-Tukey algorithms (MD CT) are reviewed. The row-column algorithm will be used in chapters 3 and 4 as a comparative measure. The MD CT algorithm will be used to develop the hybrid algorithm of chapter 4. Reduced transform algorithms are discussed, and a parametric line formulation of the linear congruence algorithm is described.

MD DFT Definition

The summation form of the multidimensional discrete Fourier transform is given by

$$Y(j_1, j_2, \ldots, j_d) = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \cdots \sum_{k_d=0}^{n_d-1} x(k_1, k_2, \ldots, k_d) \omega_{n_1}^{j_1 k_1} \omega_{n_2}^{j_2 k_2} \cdots \omega_{n_d}^{j_d k_d} \qquad (1.1)$$

$$0 \leq j_i, k_i < n_i, \quad \omega_{n_i} = e^{\frac{-2\pi\sqrt{-1}}{n_i}}.$$

Letting $n_i = N$, the direct computation of a single output point by the summation definition or by matrix multiplication requires N^d complex multiplies, and N^d complex additions. There are N^d output points, so the evaluation of the entire expression by this method requires $N^d N^d$ complex multiplies, and $N^d N^d$ complex additions. As in the 1D case fast algorithms exist for MD DFT. The row-column algorithm below is one of these.

Row-Column Algorithms

The row-column algorithm evaluates the $n \times n$ 2D DFT by computing the 1D $n-$point DFT of the rows and columns of the 2D array. The $n \times n$ 2D DFT summation is defined as

$$Y(j_1, j_2) = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} x(k_1, k_2) \omega_{n_1}^{j_1 k_1} \omega_{n_2}^{j_2 k_2}, \qquad (1.2)$$

Equation (1.2) is seen to compute one dimensional $n-$point DFT on the rows then columns of the input array. The row-column method is extended to any number of dimensions by performing 1D DFT with respect to each dimension.

There are dN^{d-1} one dimensional Fourier transforms required to evaluate the $d-$dimensional Fourier transform by the row-column method. If they are computed directly then $O(dN^{d+1})$ arithmetic operations are required. If $N = 2^n$ and they are computed

using the FFT, then the complexity is $O(dN^d \log N)$. This represents a considerable improvement over the $O(N^{2d})$ operations required for direct evaluation of (1.2).

Multidimensional Cooley-Tukey Algorithms

The row-column method applies the Cooley-Tukey (CT) FFT algorithm to each dimension separately. For the 2D case the CT FFT is applied row-wise then column-wise. This decimates and transforms the array in each index separately. In [14] a derivation of an algorithm that decimates and transforms both indices simultaneously is given. This algorithm is a multidimensional Cooley-Tukey algorithm in the sense that it exploits the additive properties of the underlying indexing set to decompose the MD DFT into stages of lower order DFT, stages of permutation, and stages of twiddle multiplication. Below, the multidimensional Cooley-Tukey algorithm is shown for the 2D case and the MD case.

The $N_1 \times N_2$ 2D DFT is given by

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) \omega_{N_1}^{k_1 n_1} \omega_{N_2}^{k_2 n_2}. \tag{1.3}$$

For N_1 and N_2 composite, $N_1 = r_1 s_1$ and $N_2 = r_2 s_2$ let

$$\begin{aligned} n_1 &= s_1 p_1 + u_1, \quad n_2 = s_2 p_2 + u_2 \\ 0 &\le p_i < r_i, \quad 0 \le u_i < s_i. \end{aligned} \tag{1.4}$$

and let

$$\begin{aligned} k_1 &= a_1 + r_1 b_1, \quad k_2 = a_2 + r_2 b_2 \\ 0 &\le a_i < r_i, \quad 0 \le b_i < s_i. \end{aligned} \tag{1.5}$$

Substituting (1.5) and (1.4 into (1.3) gives the two-dimensional Cooley-Tukey (vector-radix) decomposition [14]

$$\begin{aligned} Y(a_1 + r_1 b_1, a_2 + r_2 b_2) \;=\; & \sum_{u_1=0}^{s_1-1} \sum_{u_2=0}^{s_2-1} \omega_{s_1}^{u_1 b_1} \omega_{s_2}^{u_2 b_2} \omega_{N_1}^{a_1 u_1} \omega_{N_2}^{a_2 u_2} \\ & \cdot \sum_{p_1=0}^{r_1-1} \sum_{p_2=0}^{r_2-1} x(s_1 p_1 + u_1, s_2 p_2 + u_2) \omega_{r_1}^{a_1 p_1} \omega_{r_2}^{a_2 p_2} \end{aligned} \tag{1.6}$$

The decomposition of (1.7) extends naturally to multidimensional transforms that have a composite number of points in each dimension. Let

$$N_i = s_i r_i, \; k_i = a_i + r_i b_i, \; \text{and } n_i = s_i p_i + u_i \tag{1.7}$$

where

$$0 \le a_i, p_i < r_i, \quad 0 \le b_i, u_i < s_i, \quad i = 1, 2, \ldots, d$$

then the $d-$dimensional DFT given by (1.1) can be written

$$Y(a_1 + r_1 b_1, \ldots, a_d + r_d b_d) = \tag{1.8}$$
$$\sum_{u_1=0}^{s_1-1} \cdots \sum_{u_d=0}^{s_d-1} \omega_{s_1}^{u_1 b_1} \cdots \omega_{s_d}^{u_d b_d} \omega_{N_1}^{a_1 u_1} \cdots \omega_{N_d}^{a_d u_d}$$
$$\cdot \sum_{p_1=0}^{r_1-1} \cdots \sum_{p_d=0}^{r_d-1} x(s_1 p_1 + u_1, \ldots, s_d p_d + u_d) \omega_{r_1}^{a_1 p_1} \cdots \omega_{r_d}^{a_d p_d}$$

In chapter 4 this algorithm will be coupled with the hyperplane algorithm to produce a multiprocessor algorithm that is highly scalable and requires no interprocessor communication.

1.3 Reduced Transform Algorithms

Reduced transform algorithms (RTA) take their name from the fact that they reduce the number of one-dimensional DFT needed to compute the MD DFT below the dN^{d-1} required by the row-column method. These algorithms apply number theory or polynomial theory to the computation of the MD DFT. They include the Nussbaumer–Quandalle polynomial transform algorithm [25,26], the multidimensional algorithm [3] of Auslander *et al.*, the weighted redundancy transform (WRT) [35,34] of Vulis and Tsai, and the linear congruence algorithm of Gertner [11].

In [3], Auslander *et al.* introduce an algorithm for the computation of the multidimensional DFT. The algorithm depends on the underlying indexing set possessing finite field properties and is therefore restricted to arrays of prime size in each dimension. The WRT algorithm [35,34] depends on ring properties of the indexing set, and operates on any $M \times N$ array. This algorithm computes the MD DFT in terms of a set of lines that cover the output array. The output lines are formed by summation from a set of 1D DFT computed on lines covering the input array. If the computation of the 1D DFT are performed in parallel, then interprocess communication must occur to form the output lines.

The Nussbaumer-Quandalle algorithm [26,25] uses polynomial transforms to map the multidimensional DFT into a set of independent 1D DFT, and the linear congruence algorithm of [11] computes the 2D DFT by a set of independent 1D DFT along linear congruences over the indexing set. Both algorithms operate on 2D square arrays of prime, power of prime, and product of prime sizes in each dimension. Both of these algorithms evaluate the 2D DFT by a set of independent 1D DFT computed on data derived from the input set using only additions. Parallel algorithms like those considered in this program may be developed using these reduction algorithms.

The remainder of this section gives a parametric line formulation of the linear congruence algorithm of [11]. This formulation of the 2D DFT directly restricts computation to a set of lines covering the output grid. Correspondence between the Nussbaumer–Quandalle algorithm and the line algorithm are given in [1,28].

The Parametric Form of the Linear Congruence Algorithm

The restriction of the 2D DFT to a set of lines covering the output grid is described using the parametric form of the line [1]. Only the prime case of the algorithm is presented here. This formulation is computationally identical to the linear congruence algorithm in [11].

The $P \times P$ 2D DFT will be computed by restricting it to a set of lines over the indexing set $Z/P \times Z/P$. In general, the line through a point $\underline{a} = (a_1, a_2) \in Z/P \times Z/P$ is defined as the (cyclic) subgroup generated by $\underline{a}$

$$L(\underline{a}) = L((a_1, a_2)) = \{t\underline{a} = (a_1 t, a_2 t) : t = 0, 1, \ldots, P-1\}. \tag{1.9}$$

The following theorem gives a set of P point radial lines that cover $Z/P \times Z/P$ [1].

Theorem 1.3.1 *The set of $P+1$ lines*

$$\begin{array}{ll} L((m,1)), & m = 0, 1, \ldots, p-1 \\ L((1,0)) & \end{array}$$

cover $Z/P \times Z/P$.

The $P \times P$ 2D DFT can be computed by restricting it to lines covering the $P \times P$ output grid. The points on the lines $L((m,1))$ and $L((1,0))$ are given by the preaddition operations (derivation in chapter 2)

$$a_d^{(m,1)} = \sum_{i=0}^{P-1} x(i, d - mi) \tag{1.10}$$

$$a_d^{(1,0)} = \sum_{i=0}^{P-1} x(d, i) \tag{1.11}$$

and the one dimensional $P-$point DFT

$$V(mt, t) = \sum_{d=0}^{P-1} \omega_P^{dt} a_d^{(m,1)}, \qquad m = 0, 1, \ldots, P-1 \tag{1.12}$$

$$V(t, 0) = \sum_{d=0}^{P-1} \omega_P^{dt} a_d^{(1,0)}, \qquad t = 0, 1, \ldots, P-1 \tag{1.13}$$

Equations (1.12) and (1.13) evaluate the $P \times P$ 2D Fourier transform with $P+1$ one dimensional Fourier transforms on data derived from the input by the preadditions of (1.10) and (1.11).

In chapter 2 this algorithm is generalized to permit the computation of multidimensional discrete Fourier transform restricted to hyperplanes. The generalized algorithm is used in the design of multiprocessor algorithms in chapters 3 and 4.

1.4 *Report Organization*

This report is organized as follows: Algorithms for MD DFT computation were reviewed in 1.2. The hyperplane algorithm required by this program is developed in chapter 2. A direct mapping of this algorithm to the broadcast mode multiprocessor is given in chapter 3. The resulting multiprocessor algorithm requires no interprocessor communication, but does not scale flexibly. A hybrid algorithm that marries MD CT methods with hyperplane methods is given in chapter 4. Like the direct mapping algorithm, the hybrid algorithm requires no interprocessor communications. It also has the further benefit of being highly scalable. The results of this program are summarized in the final chapter.

Chapter 2

An Algorithm for the Computation of the MD DFT on Hyperplanes

Introduction

The multiprocessor algorithms developed in this program depend on the reduced transform algorithm (RTA) presented in this chapter. The algorithm computes a $d-$dimensional DFT by a set of independent $k-$dimensional DFT; it is a reduction algorithm in the sense that it has lowered the dimension of the Fourier transforms being computed. The $k-$dimensional DFT are performed on data derived from the input data using only additions, and produce $k-$dimensional hyperplanes of the output.

The algorithm is derived by restricting the $d-$dimensional DFT to a collection of subgroups of its output indexing set. In order to describe these subgroups in a natural manner, the notion of the discrete $k-$dimensional hyperplane is employed. In terms of these hyperplanes the development of the algorithm is undertaken in two parts. The first part defines a minimal set of $k-$dimensional hyperplanes that cover the $d-$dimensional array. The second part restricts the $d-$dimensional DFT to each of the $k-$dimensional hyperplanes of the covering set. The restrictions are shown to evaluate as independent $k-$dimensional DFT.

The algorithm is computationally similar to the reduced transform algorithms of Auslander *etal.* [3], Nussbaumer and Quandalle [26,25], and Gertner and Tolimieri [11,12]. Like those algorithms it computes the multidimensional DFT by a preaddition stage followed by a stage of independent DFT. The approach presented here computes the MD DFT on $k-$dimensional hyperplanes of the output array. It is a generalization and extension of the linear congruence algorithm of [11,12]. A correspondence between the line and polynomial algorithms is given in [1,28].

The motivation for developing this algorithm is the generation of a parallel algorithm for MD DFT computation that permits flexible scaling to the degree of parallelism of the target architecture. In later work a broadcast mode multiprocessor algorithm based on the algorithm presented here will be demonstrated. The main features of that algorithm are that it (1) requires no interprocessor communication, and (2) that it scales to match the degree of parallelism of the target processor.

The remainder of the presentation is organized as follows: first the definitions pertinent to the generation of $k-$dimensional hyperplanes are stated, then a minimal set

of covering $k-$dimensional hyperplanes is derived for the cases: where the array is of equal and prime size in at least $d-k+1$ dimensions, and where the array is of equal and power of prime (including 2) size in at least $d-k+1$ dimensions. For each case the restriction of the $d-$dimensional DFT to the covering set of $k-$dimensional hyperplanes is given. Both cases are shown to reduce the computation to a set of independent DFT performed on data derived from the input data using only additions. An appendix is provided that enumerates 3D and 4D cases of the algorithm.

2.1 Definitions

This section presents definitions pertinent to the derivation of the algorithm. The algorithm restricts the computation of the $d-$dimensional DFT to a collection of subgroups of the output array. These subgroups are defined on the underlying indexing set of the $d-$dimensional DFT.

The index set of the $N_1 \times \cdots \times N_d$ $d-$dimensional DFT is associated with the ring

$$\mathcal{C} = Z/N_1 \times \cdots \times Z/N_d,$$

where Z/N_i is the ring of integers modulo N_i. This set defines the region of support of the function. A point in the index set is taken to be the vector $\underline{u} \in \mathcal{C}$ defined by the $d-$tuple

$$\underline{u} = (u_1, \ldots, u_d), \quad u_i \in Z/N_i.$$

The value of the DFT restricted to a point $\underline{u}$ in the output array is evaluated in the natural way as

$$V(u_1, \ldots, u_d) = \sum_{a_1=0}^{N_1-1} \cdots \sum_{a_d=0}^{N_d-1} x(a_1, \ldots, a_d) \omega_{N_1}^{a_1 u_1} \cdots \omega_{N_d}^{a_d u_d}$$

for a fixed $\underline{u} \in \mathcal{C}$.

The discrete line through a point in a two-dimensional array $\underline{a} = (a_1, a_2) \in Z/N \times Z/N$ is defined as the (cyclic) subgroup additively generated by $\underline{a}$

$$L(\underline{a}) = \{s\underline{a} = (sa_1, sa_2) : s \in Z/N\}. \tag{2.1}$$

The line is said to be of full order if $|L(\underline{a})| = N$.

This definition of a line extends naturally from the case where the line lies in a two-dimensional grid to the case where the line lies in a $d-$dimensional array. The line through a point $\underline{a} \in (Z/N)^d$ is defined by the subgroup [12]

$$L(\underline{a}) = \{s\underline{a} = (sa_1, \ldots, sa_d) : s \in Z/N\}. \tag{2.2}$$

By definition the line $L(\underline{a})$ is additively generated by $\underline{a}$; $L((\underline{a}))$ contains all linear combinations of the point $\underline{a} \in (Z/N)^d$. $L(\underline{a})$ defines the additive closure of $\underline{a} \in (Z/N)^d$.

In a manner similar to the way $L(\underline{a})$ defines the closure of a point $\underline{a} \in (Z/N)^d$, we may define the closure of a set of points $\underline{a}_1, \ldots, \underline{a}_k \in (Z/N)^d$. The closure of $\underline{a}_1, \ldots, \underline{a}_k \in (Z/N)^d$ is given by the subgroup

$$H(\underline{a}_1, \ldots, \underline{a}_k) = \{s_1\underline{a}_1 + \cdots + s_k\underline{a}_k : s_i \in Z/N\}. \tag{2.3}$$

$H(\underline{a}_1, \ldots, \underline{a}_k)$ contains all linear combinations of the points in the set $\{\underline{a}_1, \ldots, \underline{a}_k\}$.

The idea of dimension can be associated with the subgroups generated in this way. The dimension of a subgroup of $(Z/N)^d$ is taken to be the number of linearly independent points $\underline{a}_i \in (Z/N)^d$ required to generate it by the definition of (2.3). In this manner, subgroups generated by the additive closure of 1, 2, and 3 linearly independent points will be called lines, planes, and cubes respectively. Similarly, a subgroup whose dimension is $d-1$ is called a hyperplane. If the dimension of a subgroup is $k < d$ it is called a $k-$dimensional hyperplane.

The requirement of (2.3) that the array be of equal size in each dimension is overly restrictive for the development of the algorithm. It will be seen that it is only necessary for the array to be of equal size in $d-k+1$ dimensions.

Consider a $d-$dimensional array whose region of support is given by $C = Z/N_1 \times \cdots \times Z/N_d$. Assume that $d-k+1$ dimensions are of equal size, and for simplicity of presentation allow those dimensions to be contiguous. Write $N_k = \cdots = N_d$. The definition adopted for the presentation of the algorithm uses the subgroup $H_k(\underline{a})$ formed by the closure of the vector $\underline{a} \in R$ with the $k-1$ standard basis vectors

$$\delta_i(j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}, \qquad i = 1, \ldots, k-1. \tag{2.4}$$

The subgroup

$$H_k(\underline{a}) = \{s_1\underline{\delta}_1 + \cdots + s_{k-1}\underline{\delta}_{k-1} + s_k\underline{a}_k : s_i \in Z/N_i\} \tag{2.5}$$

will define the $k-$dimensional discrete hyperplane. For the purpose of the development of the algorithm we will only be concerned with $k-$dimensional hyperplanes of full order. That is $(N_1 \cdots N_k)$ points. As an example, consider the 4D array $(Z/N)^4$. The hyperplane generated by the point $\underline{a} = (0,0,1,1)$ is given by

$$H_3((0,0,1,1)) = \{(s_1, s_2, s_3, s_3) : s_i \in Z/N\},$$

and $H_3((0,0,1,1))$ contains N^3 points. Similarly, the hyperplane generated by the point $\underline{a} = (0,0,x,y)$ is given by

$$H_3((0,0,x,y)) = \{(s_1, s_2, s_3x, s_3y) : s_i \in Z/N\}.$$

A set of $k-$dimensional hyperplanes is said to *cover* the $d-$dimensional array C, if every point in C is in at least one $k-$dimensional hyperplane of the set. A set of covering $k-$dimensional hyperplanes is *minimal* if there is no smaller set of $k-$dimensional hyperplanes that also covers C.

2.2 Prime Case

This section derives the prime case of the algorithm. For this case the $d-$dimensional DFT is defined over the region $\mathcal{C} = Z/N_1 \times \cdots \times Z/N_d$ where $\mathcal{C}$ is of equal and prime size in $d-k+1$ dimensions. That is,

$$\begin{aligned} &\mathcal{C} = Z/N_1 \times \cdots \times Z/N_d = \mathcal{A} \times \mathcal{B}, \\ &\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P, \\ &\mathcal{B} = (Z/P)^{d-k}, \text{ and } P \text{ is a prime.} \end{aligned} \tag{2.6}$$

The algorithm is derived in two parts. The first part is the specification of a minimal set of $k-$dimensional hyperplanes that cover $\mathcal{C}$. This follows immediately below. The second part of the derivation requires the $d-$dimensional DFT to be restricted to each of the $k-$dimensional hyperplanes of the covering set. The resulting algorithm is shown to be computed by a set of independent $k-$dimensional DFT. This case of the algorithm is specified by the procedure of equations (2.20) and (2.21).

Covering Hyperplanes

The following theorem gives a minimal set of $|\mathcal{A}|$ point $k-$dimensional hyperplanes that cover R. Let $\underline{\theta}(j)$ denote the vector of j zeros, ie. $\underline{\theta}(3) = (0,0,0)$. The $k-$dimensional hyperplanes are defined by equation (2.5).

Theorem 2.2.1 *The set of $k-$dimensional hyperplanes*

$$\begin{aligned} \mathcal{H}_0 &= H_k((\underline{\theta}_{(k-1)}, 1, m_1, \ldots, m_{d-k})) \qquad m_i \in Z/P \\ \mathcal{H}_1 &= H_k((\underline{\theta}_{(k-1)}, 0, 1, m_2, \ldots, m_{d-k})) \\ &\vdots \\ \mathcal{H}_{d-k} &= H_k((\underline{\theta}_{(k-1)}, 0, 0, \ldots, 0, 1)) \end{aligned}$$

covers the $d-$dimensional array $\mathcal{C} = \mathcal{A} \times \mathcal{B}$, where $\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P$, $\mathcal{B} = (Z/P)^{d-k}$, and P is a prime. There are

$$\frac{P^{d-k+1}-1}{P-1}$$

$k-$dimensional hyperplanes in the set.

Proof. Every $\underline{a} = (a_1, \ldots, a_d) \in \mathcal{C}$ can be written

$$\underline{a} = \underline{a}_{(1)} + \cdots + \underline{a}_{(k-1)} + \underline{a}',$$

where $\underline{a}_{(i)}$ is the vector whose i^{th} component is a_i and is zero elsewhere, and $\underline{a}'$ is the vector $\underline{a}' = (\underline{\theta}_{(k-1)}, a_k, \ldots, a_d)$.

By definition of $H_k(\underline{a})$, $\underline{a}_{(i)}$ is in every hyperplane of the covering set for $i = 1, \ldots, k-1$. Since the hyperplanes are closed under addition, covering is shown if every $\underline{a}'$ is in

at least one hyperplane of the covering set. If $a_k \neq 0$ then a_k is invertible modulo P and $\underline{a}'$ may be rewritten

$$\underline{a}' = a_k(\underline{\theta}_{(k-1)}, 1, a_k^{-1}a_{k+1}, \ldots, a_k^{-1}a_d),$$

and $\underline{a}'$ is seen to be contained in the union of the hyperplanes of the set $\mathcal{H}_0$ of the theorem. By closure, the point $\underline{a} = \underline{a}^1 + \cdots + \underline{a}^{k-1} + \underline{a}'$ must also be contained in the union of the hyperplanes of the set $\mathcal{H}_0$. There are P^{d-k} $k-$dimensional hyperplanes in $\mathcal{H}_0$.

Repetition of this step until all the remaining $d-k$ components of $\underline{a}'$ are exhausted completes the proof. The α^{th} set of hyperplanes generated contains $P^{d-k-\alpha}$ $k-$dimensional hyperplanes. In all there are

$$\frac{P^{d-k+1}-1}{P-1}$$

$k-$dimensional hyperplanes in the covering set.
□

The $k-$dimensional hyperplanes of the covering set of theorem 2.2.1 contain redundant points. The number of points in $\mathcal{C}$ is

$$|\mathcal{C}| = N_1 \cdots N_{k-1} \cdot P^{d-k+1}.$$

The number of points in each $k-$dimensional hyperplane is

$$|\mathcal{A}| = N_1 \cdots N_{k-1} \cdot P,$$

and the number of redundant points is

$$|\mathcal{A}| \cdot \left(\frac{P^{d-k}-1}{P-1}\right).$$

The locations of the redundant points are given by the $(k-1)$ dimensional hyperplane

$$H(\underline{\delta}_1, \ldots, \underline{\delta}_{k-1}) = \{(s_1, \ldots, s_{k-1}, \underline{\theta}(d-k+1)) : s_i \in Z/N_i\} \tag{2.7}$$

where $H(\underline{\delta}_1, \ldots, \underline{\delta}_{k-1})$ is defined by equation (2.3). The elements of (2.7) are common to every hyperplane of theorem 2.2.1, and are redundant

$$P \cdot \left(\frac{P^{d-k}-1}{P-1}\right)$$

times, accounting for all redundant points.

DFT Restriction

The $d-$dimensional DFT can be computed by restricting it to the $k-$dimensional hyperplanes of theorem 2.2.1. Below, the restriction of the MD DFT to those hyperplanes is derived.

The $d-$dimensional DFT over $\mathcal{C} = Z/N_1 \times \cdots Z/N_d$ is given by

$$V(\underline{u}) = \sum_{\underline{a} \in \mathcal{C}} x(\underline{a}) \omega_{N_1}^{a_1 u_1} \cdots \omega_{N_d}^{a_d u_d}, \qquad \underline{u} \in \mathcal{C}. \tag{2.8}$$

The prime case of the algorithm requires $d-k+1$ dimensions to be of equal and prime size. For simplicity of presentation assume these are contiguous and write

$$\begin{aligned} &\mathcal{C} = \mathcal{A} \times \mathcal{B}, \\ &\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times P, \\ &\mathcal{B} = (Z/P)^{d-k}, \text{ and } P \text{ is a prime.} \end{aligned}$$

The P^{d-k} $k-$dimensional hyperplanes of $\mathcal{H}_0$ given by theorem 2.2.1 can be defined by the homomorphism

$$\Phi_0 : \mathcal{A} \longmapsto \mathcal{C}. \tag{2.9}$$

For all $\underline{s} \in \mathcal{A}$, Φ_0 is given by

$$\Phi_0(\underline{s}) = (s_1, \ldots, s_{k-1}, s_k, s_k m_1, \ldots, s_k m_{d-k}). \tag{2.10}$$

The restriction of the $d-$dimensional DFT (2.8) to the $k-$dimensional hyperplanes of $\mathcal{H}_0$ is evaluated by replacing $\underline{u}$ with $\Phi_0(\underline{s})$ of (2.10) to obtain

$$V\left(\Phi_0(\underline{s})\right) = \sum_{\underline{a} \in \mathcal{C}} x(\underline{a}) \omega_{N_1}^{s_1 a_1} \cdots \omega_{N_d}^{s_k m_{d-k} a_d}, \tag{2.11}$$

$$\underline{s} \in \mathcal{A}.$$

The inner product

$$\Phi_0(\underline{s}) \cdot \underline{a} = s_1 a_1 + \cdots + s_{k-1} a_{k-1} + s_k (a_k + a_{k+1} m_1 + \cdots + a_d m_{d-k}) \tag{2.12}$$

can be written

$$\Phi_0(\underline{s}) \cdot \underline{a} = s_1 d_1 + \cdots + s_k d_k = \underline{s} \cdot \underline{d} \tag{2.13}$$

$$\underline{d} \in \mathcal{A}$$

where

$$\begin{aligned} &d_j = a_j, \qquad j = 1, \ldots, k-1 \\ &i_{j-k} = a_j, \qquad j = k+1, \ldots, d \\ &d_k = a_k + m_1 a_{k+1} + \cdots + m_{d-k} a_d \ . \end{aligned} \tag{2.14}$$

Substituting (2.13) in (2.11) allows the $d-$dimensional DFT restricted to the $k-$dimensional hyperplanes of $\mathcal{H}_0$, to be written

$$V(\Phi_0(\underline{s})) = \sum_{\underline{d}\in\mathcal{A}} \sum_{\underline{i}\in\mathcal{B}} x(\underline{d}_{(k-1)},\, d_k - \underline{m}\cdot\underline{i},\, \underline{i})\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}, \tag{2.15}$$

$$\underline{d}\in\mathcal{A}.$$

Where

$$a_k = d_k - \underline{m}\cdot\underline{i}, \tag{2.16}$$

and

$$\underline{d}_{(k-c)} = (d_1,\ldots,d_{k-c}). \tag{2.17}$$

Equation (2.15) can be rewritten as a reduction stage, followed by a DFT stage. The reduction operation is given by

$$a_{\underline{d}}^{\mathcal{H}_0} = \sum_{\underline{i}\in\mathcal{B}} x\left(\underline{d}_{(k-1)},\, d_k - \underline{m}\cdot\underline{i},\, \underline{i}\right), \quad \underline{d}\in\mathcal{A} \tag{2.18}$$

and the DFT stage is given by

$$V(\Phi_0(\underline{s})) = \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{\mathcal{H}_0}\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}, \tag{2.19}$$

$$\underline{s}\in\mathcal{A}, \quad \underline{m}\in\mathcal{B}.$$

Equations (2.18) and (2.19) give the values of the $d-$dimensional DFT on the $k-$dimensional hyperplanes of the set $\mathcal{H}_0$. There is one such hyperplane for every $\underline{m}\in\mathcal{B}$.

In a similar manner, the restriction of the $d-$dimensional DFT to each of the remaining $k-$dimensional hyperplanes of the covering set of theorem 2.2.1 is computed by a reduction operation $a_{\underline{d}}^{\mathcal{H}_\alpha}$, followed by a $k-$dimensional DFT of those points.

The complete algorithm is stated below for the case where the $d-$dimensional DFT over $\mathcal{C}$ is of prime size in at least $d-k+1$ dimensions. Then $\mathcal{C}$ is given by

$$\begin{aligned}&\mathcal{C} = \mathcal{A}\times\mathcal{B},\\ &\mathcal{A} = N_1\times\cdots\times N_{k-1}\times P,\\ &\mathcal{B} = (Z/P)^{d-k}, \text{ and } P \text{ is a prime.}\end{aligned}$$

Step 1 Reduction stage. This stage requires the evaluation of the summations

$$\begin{aligned}
a_{\underline{d}}^{\mathcal{H}_0} &= \sum_{\underline{i}\in\mathcal{B}} x(\underline{d}_{(k-1)},\, d_k - \sum_{l=1}^{d-k} m_l i_l,\, \underline{i}), \\
&\vdots \\
a_{\underline{d}}^{\mathcal{H}_j} &= \sum_{\underline{i}\in\mathcal{B}} x(\underline{d}_{(k-1)},\, \underline{i}_{(j)},\, d_k - \sum_{l=j+1}^{d-k} m_l i_l,\, i_{j+1},\ldots,i_{d-k}), \\
&\vdots \\
a_{\underline{d}}^{\mathcal{H}_{d-k}} &= \sum_{\underline{i}\in\mathcal{B}} x(\underline{d}_{(k-1)},\, \underline{i},\, d_k),
\end{aligned} \tag{2.20}$$

$$\underline{d} \in \mathcal{A}, \quad m_i \in Z/N_i$$

Step 2 DFT stage. This stage requires the evaluation of the $k-$dimensional DFT

$$\begin{aligned} V(\Phi_0(\underline{s})) &= \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{\mathcal{H}_0} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k}, \qquad (2.21) \\ &\vdots \\ V(\Phi_j(\underline{s})) &= \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{\mathcal{H}_j} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k}, \\ &\vdots \\ V(\Phi_{d-k}(\underline{s})) &= \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{\mathcal{H}_{d-k}} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k}, \\ & \\ & \underline{s} \in \mathcal{A}. \end{aligned}$$

In general Φ_j is given by

$$\Phi_j(\underline{s}) = (s_1, \ldots, s_{k-1}, \underline{\theta}(j), s_k, s_k m_{j+1}, \ldots, s_k m_{d-k}),$$

$$m_i \in Z/N_i.$$

The $d-$dimensional DFT over $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ is seen to be computed by

$$\frac{P^{d-k+1} - 1}{P - 1}$$

independent $k-$dimensional DFT over $\mathcal{A}$. The $k-$dimensional DFT are evaluated on data derived from the input data using only additions.

2.3 *Power of Prime Case*

This section derives the power of prime case of the algorithm. Like the prime case, the power of prime case is developed in two parts. The first part is the specification of a minimal set of covering $k-$dimensional hyperplanes. The second part is the restriction of the $d-$dimensional DFT to each of the hyperplanes of the covering set. The derivation of the algorithm follows below. The algorithm is stated by the procedure of equations (2.34) and (2.35).

Covering Hyperplanes

Consider the $d-$dimensional DFT defined over

$$\mathcal{C} = Z/N_1 \times \cdots \times Z/N_d$$

where $\mathcal{C}$ is of equal and power of prime size in at least $d-k+1$ dimensions. For ease of presentation these dimensions are taken to be contiguous. That is,

$$\begin{aligned} &\mathcal{C} = \mathcal{A} \times \mathcal{B}, \\ &\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P^n, \\ &\mathcal{B} = (Z/P^n)^{d-k}, \text{ and } P \text{ is any prime including 2.} \end{aligned} \tag{2.22}$$

The following theorem gives a minimal set of $|\mathcal{A}|$ point $k-$dimensional hyperplanes that cover $\mathcal{C}$. Let $\underline{\theta}_{(j)}$ denote the vector of j zeros, ie. $\underline{\theta}_{(3)} = (0,0,0)$. The $k-$dimensional hyperplanes are defined by equation (2.5).

Theorem 2.3.1 *The set of $k-$dimensional hyperplanes*

$$\begin{aligned} \mathcal{H}_0 &= H_k((\underline{\theta}_{(k-1)}, 1, m_1, \ldots, m_{d-k})), & m_i \in Z/P^n \\ \mathcal{H}_1 &= H_k((\underline{\theta}_{(k-1)}, r_1 P, 1, m_2, \ldots, m_{d-k})), & r_i \in Z/P^{n-1} \\ &\vdots \\ \mathcal{H}_{d-k} &= H_k((\underline{\theta}_{(k-1)}, Pr_1, Pr_2, \ldots, Pr_{d-k}, 1)) \end{aligned}$$

covers the $d-$dimensional array $\mathcal{C} = \mathcal{A} \times \mathcal{B}$, where $\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P^n$, $\mathcal{B} = (Z/P^n)^{d-k}$, and P is any prime including 2. There are

$$\left(\frac{P^n}{P}\right)^{d-k} \cdot \left(\frac{P^{d-k+1}-1}{P-1}\right)$$

$k-$dimensional hyperplanes in the set.

Proof. Similar to theorem 2.2.1.
□

The covering set of theorem 2.3.1 contains redundant points. Altogether there are

$$|\mathcal{A}| \left(\frac{P^n}{P}\right)^{d-k} \left(\frac{P^{d-k}-1}{P-1}\right)$$

such points. The following theorems fully describe the redundancy between any two hyperplanes of the covering set of theorem 2.3.1. Throughout the sequel, the notation $\underline{x}_{\bmod y}$ is taken to be

$$\underline{x}_{\bmod y} = (x_{1 \bmod y}, \ldots, x_{n \bmod y})$$

where $\underline{x}$ is the vector $\underline{x} = (x_1 \ldots, x_n)$ and y is a scalar.

Theorem 2.3.2 *The redundancy between any two $k-$dimensional hyperplanes of the set $\mathcal{H}_0$ of theorem 2.3.1 is given by*

$$\begin{aligned} H_k((\underline{\theta}_{(k-1)}, 1, \underline{m})) \cap H_k((\underline{\theta}_{(k-1)}, 1, \underline{j})) = \\ H_k((\underline{\theta}_{(k-1)}, P^{n-\alpha}, (\underline{m}_{\bmod P^\alpha})P^{n-\alpha})), \quad m_i, j_i \in Z/P^n. \end{aligned}$$

Where α is the maximum such that

$$m_i \equiv j_i \bmod P^\alpha$$

holds for all $i = 1, \ldots, d-k$. The intersection contains $(N_1 \cdots N_{k-1})P^\alpha$ points.

Theorem 2.3.3 *The redundancy between any two $k-$dimensional hyperplanes of the set $\mathcal{H}_l$, $l = 1, \ldots, d-k-1$, of theorem 2.3.1 is given by*

$$H_k((\underline{\theta}_{(k-1)}, \underline{r}, 1, \underline{m})) \cap H_k((\underline{\theta}_{(k-1)}, \underline{q}, 1, \underline{j})) = H_k((\underline{\theta}_{(k-1)}, P^{n-\alpha+1}(\underline{r}_{\bmod P^\alpha}), P^{n-\alpha}, P^{n-\alpha}(\underline{m}_{\bmod P^\alpha})))$$

where

$$r_i, q_i \in Z/P^{n-1}, \quad i = 1, \ldots, l$$
$$m_i, j_i \in Z/P^n, \quad i = l+1, \ldots, d-k$$

and α is the maximum such that

$$r_i \equiv q_i \bmod P^\alpha$$

and

$$m_h \equiv j_h \bmod P^\alpha$$

hold for all $i = 1, \ldots, l$ and $h = l+1, \ldots, d-k$. The intersection contains $(N_1 \cdots N_{k-1})P^\alpha$ points.

Theorem 2.3.4 *The redundancy between any two $k-$dimensional hyperplanes of the set $\mathcal{H}_{d-k}$ of theorem 2.3.1 is given by*

$$H_k((\underline{\theta}_{(k-1)}, P\underline{r}, 1)) \cap H_k((\underline{\theta}_{(k-1)}, P\underline{q}, 1)) = H_k((\underline{\theta}_{(k-1)}, P^{n-\alpha}(\underline{r}_{\bmod P^\alpha}), P^{n-\alpha-1}, P^{n-\alpha}(\underline{m}_{\bmod P^\alpha})))$$

where

$$r_i, q_i \in Z/P^{n-1}$$

and α is the maximum such that

$$r_i \equiv q_i \bmod P^\alpha$$

holds for all $i = 1, \ldots, d-k$. The intersection contains $(N_1 \cdots N_{k-1})P^\alpha$ points.

Theorem 2.3.5 *The redundancy between any two $k-$dimensional hyperplanes of the sets $\mathcal{H}_x$ and $\mathcal{H}_y$, where $x \neq y$ is given by*

$$H_k((\underline{\theta}_{(k-1)}, \underline{r}, 1, \underline{m})) \cap H_k((\underline{\theta}_{(k-1)}, \underline{q}, 1, \underline{j})) = H_k(\underline{\delta}_1, \ldots, \underline{\delta}_{k-1}, \underline{\theta}_{(d-k+1)}).$$

The intersection contains $(N_1 \cdots N_{k-1})$ points.

DFT Restriction

In a manner similar to the prime case described in the previous section, the $d-$dimensional DFT can be computed by restricting it to the covering set of $k-$dimensional hyperplanes given by theorem 2.3.1. This restriction is detailed below.

The power of prime case of the algorithm requires $d-k+1$ dimensions to be of equal and power of prime size (including 2). For simplicity of presentation these are assumed to be contiguous, and we write

$$\begin{aligned}
&\mathcal{C} = \mathcal{A} \times \mathcal{B}, \\
&\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P^n, \\
&\mathcal{B} = (Z/P^n)^{d-k}, \text{ and } P \text{ is any prime (including 2).}
\end{aligned}$$

The restriction of the DFT to the $(P^n)^{d-k}$ $k-$dimensional hyperplanes of the set $\mathcal{H}_0$ is identical to that of the prime case derived in the previous section. To illustrate the derivation of the other terms, the set $\mathcal{H}_1$ of theorem 2.3.1 is considered below.

The $(1/P)(P^n)^{d-k}$ $k-$dimensional hyperplanes of $\mathcal{H}_1$ of theorem 2.3.1 can be defined by the homomorphism

$$\Omega_1 : \mathcal{A} \longmapsto \mathcal{C}. \tag{2.23}$$

For all $\underline{s} \in \mathcal{A}$, Ω_1 is given by

$$\Omega_1(\underline{s}) = (s_1, \ldots, s_{k-1}, s_k P r_1, s_k, s_k m_2, \ldots, s_k m_{d-k}). \tag{2.24}$$

The restriction of the $d-$dimensional DFT to the $k-$dimensional hyperplanes of $\mathcal{H}_1$ is evaluated by replacing $\underline{u}$ with $\Omega_1(\underline{s})$ of (2.24) to obtain

$$V(\Omega_1(\underline{s})) = \sum_{\underline{a} \in \mathcal{C}} x(\underline{a}) \omega_{N_1}^{s_1 a_1} \cdots \omega_{N_d}^{s_k m_{d-k} a_d}, \tag{2.25}$$

$$\underline{s} \in \mathcal{A}.$$

The inner product

$$\begin{aligned}
\Omega_1(\underline{s}) \cdot \underline{a} \;=\; & s_1 a_1 + \cdots + s_{k-1} a_{k-1} \\
& + s_k (a_k r_1 P + a_{k+1} + a_{k+2} m_2 + \cdots + a_d m_{d-k})
\end{aligned} \tag{2.26}$$

gives

$$\Omega_1(\underline{s}) \cdot \underline{a} = s_1 d_1 + \cdots + s_k d_k = \underline{s} \cdot \underline{d} \tag{2.27}$$

$$\underline{d} \in \mathcal{A}$$

where

$$\begin{aligned}
&d_j = a_j, \qquad j = 1, \ldots, k-1 \\
&i_1 = a_k, \\
&i_{j-k} = a_j, \qquad j = k+2, \ldots, d \\
&d_k = a_k r_1 P + a_{k+1} + a_{k+2} m_2 + \cdots + m_{d-k} a_d \; .
\end{aligned} \tag{2.28}$$

Equation (2.25) may then be written

$$V(\Omega_1(\underline{s})) = \sum_{\underline{d}\in A}\sum_{\underline{i}\in B}\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}\cdot \tag{2.29}$$

$$x(\underline{d}_{(k-1)}, i_1, d_k - Pr_1 i_1 - \sum_{l=2}^{d-k} m_l i_l, i_2, \ldots, i_{d-k}),$$

$$\underline{d}\in A.$$

Where

$$a_{k+1} = d_k - i_1 r_1 P - \sum_{l=2}^{d-k} m_l i_l, \tag{2.30}$$

and

$$\underline{d}_{(k-c)} = (d_1, \ldots, d_{k-c}). \tag{2.31}$$

Equation (2.29) can be rewritten as a reduction stage, followed by a DFT stage. The reduction operation is given by

$$a_{\underline{d}}^{\mathcal{H}_1} = \sum_{\underline{i}\in\mathcal{B}} x\left(\underline{d}_{(k-1)}, i_1, d_k - i_1 r_1 P - \sum_{l=2}^{d-k} i_l m_l, i_2, \ldots, i_{d-k}\right), \quad \underline{d}\in\mathcal{A} \tag{2.32}$$

and the DFT stage is given by

$$V(\Omega_1(\underline{s})) = \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{\mathcal{H}_1}\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}, \tag{2.33}$$

$$\underline{s}\in\mathcal{A}, \quad \underline{m}\in\mathcal{B}.$$

Equations (2.32) and (2.33) give the values of the $d-$dimensional DFT on the $k-$dimensional hyperplanes of the set $\mathcal{H}_1$.

In a similar manner, the restriction of the $d-$dimensional DFT to each of the remaining $k-$dimensional hyperplanes of the covering set of theorem 2.3.1 is computed by a reduction operation $a_{\underline{d}}^{\mathcal{H}_\alpha}$, followed by a $k-$dimensional DFT of those points.

The complete algorithm is stated below for the case where the region of support, $\mathcal{C}$, is of power of prime size in at least $d-k+1$ dimensions. Let $\mathcal{C}$ be given by

$$\mathcal{C} = \mathcal{A}\times\mathcal{B},$$
$$\mathcal{A} = N_1\times\cdots\times N_{k-1}\times P^n,$$
$$\mathcal{B} = (Z/P^n)^{d-k}, \text{ and } P \text{ is a prime (including 2).}$$

For this case, the algorithm is specified by the following procedure:

Step 1 Reduction stage. This stage requires the evaluation of the summations

$$a_{\underline{d}}^{\mathcal{H}_1} = \sum_{\underline{i}\in\mathcal{B}} x\left(\underline{d}_{(k-1)}, d_k - \sum_{l=1}^{d-k} m_l i_l, \underline{i}\right), \tag{2.34}$$

$$
\begin{aligned}
&\vdots \\
a_{\underline{d}}^{\mathcal{H}_j} &= \sum_{\underline{i}\in\mathcal{B}} x\left(\underline{d}_{(k-1)},\ \underline{i}_{(j)},\ d_k - \sum_{l=1}^{j} i_l r_l P - \sum_{l=j+1}^{d-k} m_l i_l,\ i_{j+1},\ldots,i_{d-k}\right), \\
&\vdots \\
a_{\underline{d}}^{\mathcal{H}_{d-k}} &= \sum_{\underline{i}\in\mathcal{B}} x\left(\underline{d}_{(k-1)},\ \underline{i},\ d_k - \sum_{l=1}^{d-k} i_l r_l P\right)
\end{aligned}
$$

$$
\underline{d}\in\mathcal{A}, \quad m_i \in Z/N_i
$$

Step 2 DFT stage. This stage requires the evaluation of the $k-$dimensional DFT

$$
\begin{aligned}
V(\Omega_0(\underline{s})) &= \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{\mathcal{H}_0}\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}, \qquad (2.35) \\
&\vdots \\
V(\Omega_j(\underline{s})) &= \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{\mathcal{H}_j}\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}, \\
&\vdots \\
V(\Omega_{d-k}(\underline{s})) &= \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{\mathcal{H}_{d-k}}\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}, \\
&\underline{s}\in\mathcal{A}.
\end{aligned}
$$

The $d-$dimensional DFT over $\mathcal{C} = \mathcal{A}\times\mathcal{B}$ is seen to be computed by

$$
\left(\frac{P^n}{P}\right)\frac{P^{d-k+1}-1}{P-1}
$$

independent $k-$dimensional DFT over $\mathcal{A}$. The $k-$dimensional DFT are evaluated on data derived from the input data using only additions.

2.4 Appendix

This section enumerates the 3D and 4D cases of the algorithm.

- 3D $\longrightarrow$ 2D, Prime Case.
  $N \times P \times P$ 3D DFT computed by $N \times P$ 2D DFT,
  where P is prime and N is any number.

  1. Covering planes

$$\mathcal{H}_0 = H_2((0,1,m)), \quad m = 0,1,\ldots,P-1$$
$$\mathcal{H}_1 = H_2((0,0,1)).$$

  2. Reduction stage

$$a^{\mathcal{H}_0}_{(d_1,d_2)} = \sum_{i=0}^{P-1} x(d_1,\, d_2 - im,\, i), \qquad m = 0,1,\ldots,P-1$$
$$a^{\mathcal{H}_1}_{(d_1,d_2)} = \sum_{i=0}^{P-1} x(d_1,\, i,\, d_2).$$

$$d_1 = 0,1,\ldots,N-1$$
$$d_2 = 0,1,\ldots,P-1$$

  3. $N \times P$ 2D DFT stage

$$V(s_1,\, s_2,\, s_2 m) = \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{\mathcal{H}_0}_{(d_1,d_2)}\omega_N^{d_1 s_1}\omega_P^{d_2 s_2}$$
$$V(s_1,\, 0,\, s_2) = \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{\mathcal{H}_1}_{(d_1,d_2)}\omega_N^{d_1 s_1}\omega_P^{d_2 s_2}$$

$$s_1 = 0,1,\ldots,N-1$$
$$s_2 = 0,1,\ldots,P-1 \qquad m = 0,1,\ldots,P-1.$$

The computation of the $N \times P \times P$ 3D DFT by this method requires $P+1$ reduction operations and a like number of $N \times P$ 2D DFT. Note that [33] is a source of $O(P \log P)$ prime size FFT algorithms.

To compute a 3D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P \times P \times P$ 3D DFT computed as $P^2 + P + 1$ reduction operations and a like number of P point 1D DFT.

- 3D $\longrightarrow$ 2D , Power of Prime Case.
  $N \times P^n \times P^n$ 3D DFT computed by $N \times P^n$ 2D DFT,
  where P is prime (including 2) and N is any number.

  1. Covering planes

$$\begin{aligned} \mathcal{H}_0 &= H_2((0,1,m)), \quad m = 0,1,\ldots,P^n-1 \\ \mathcal{H}_1 &= H_2((0,rP,1)), \quad r = 0,1,\ldots,P^{n-1}-1. \end{aligned}$$

  2. Reduction stage

$$\begin{aligned} a^{\mathcal{H}_0}_{(d_1,d_2)} &= \sum_{i=0}^{P^n-1} x(d_1,\, d_2 - im,\, i), \qquad m = 0,1,\ldots,P^n-1 \\ a^{\mathcal{H}_1}_{(d_1,d_2)} &= \sum_{i=0}^{P^n-1} x(d_1,\, i,\, d_2 - rPi), \qquad r = 0,1,\ldots,P^{n-1}-1 \end{aligned}$$

$$\begin{aligned} d_1 &= 0,1,\ldots,N-1 \\ d_2 &= 0,1,\ldots,P^n-1 \end{aligned}$$

  3. $N \times P^n$ 2D DFT stage

$$\begin{aligned} V(s_1,\, s_2,\, s_2 m) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P^n-1} a^{\mathcal{H}_0}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_{P^n}^{d_2 s_2} \\ V(s_1,\, rPs_2,\, s_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P^n-1} a^{\mathcal{H}_1}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_{P^n}^{d_2 s_2} \end{aligned}$$

$$\begin{aligned} s_1 &= 0,1,\ldots,N-1 \qquad & m &= 0,1,\ldots,P^n-1, \\ s_2 &= 0,1,\ldots,P^n-1 \qquad & r &= 0,1,\ldots,P^{n-1}-1. \end{aligned}$$

The computation of the $N \times P^n \times P^n$ 3D DFT by this method requires $P^n + P^{n-1}$ reduction operations and a like number of 2D $N \times P^n$ DFT.

To compute a 3D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P^n \times P^n \times P^n$ DFT computed by $(P^n)^2(1 + 1/P + 1/P^2)$ reduction operations and a like number of P^n point 1D DFT.

- 4D $\longrightarrow$ 3D , Prime Case.
  $N_1 \times N_2 \times P \times P$ 4D DFT computed by $N_1 \times N_2 \times P$ 3D DFT,
  where P is prime and N_1, N_2 are any numbers.

  1. Covering cubes

$$\mathcal{H}_0 = H_3((0,0,1,m)), \quad m = 0,1,\ldots,P-1$$
$$\mathcal{H}_1 = H_3((0,0,0,1)).$$

  2. Reduction stage

$$a^{\mathcal{H}_0}_{(d_1,d_2,d_3)} = \sum_{i=0}^{P-1} x(d_1,\, d_2,\, d_3 - im,\, i), \qquad m = 0,1,\ldots,P-1$$
$$a^{\mathcal{H}_1}_{(d_1,d_2,d_3)} = \sum_{i=0}^{P-1} x(d_1,\, d_2,\, i,\, d_3)$$

$$d_1 = 0,1,\ldots,N_1-1$$
$$d_2 = 0,1,\ldots,N_2-1$$
$$d_3 = 0,1,\ldots,P-1$$

  3. $N \times N \times P$ 3D DFT stage

$$V(s_1,\, s_2,\, s_3,\, s_3 m) = \sum_{d_1=0}^{N_1-1}\sum_{d_2=0}^{N_2-1}\sum_{d_3=0}^{P-1} a^{\mathcal{H}_0}_{(d_1,d_2,d_3)}\omega_{N_1}^{d_1 s_1}\omega_{N_2}^{d_2 s_2}\omega_P^{d_3 s_3}$$
$$V(s_1,\, s_2,\, 0,\, s_3) = \sum_{d_1=0}^{N_1-1}\sum_{d_2=0}^{N_2-1}\sum_{d_3=0}^{P-1} a^{\mathcal{H}_1}_{(d_1,d_2,d_3)}\omega_{N_1}^{d_1 s_1}\omega_{N_2}^{d_2 s_2}\omega_P^{d_3 s_3}$$

$$s_1 = 0,1,\ldots,N_1-1,$$
$$s_2 = 0,1,\ldots,N_2-1, \qquad m = 0,1,\ldots,P-1$$
$$s_3 = 0,1,\ldots,P-1$$

The computation of the $N_1 \times N_2 \times P \times P$ 4D DFT by this method requires $P+1$ reduction operations and a like number of 3D $N_1 \times N_2 \times P$ DFT. Note that [33] is a source of $O(P \log P)$ prime size FFT algorithms.

To compute a 4D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P \times P \times P \times P$ DFT computed as P^3+P^2+P+1 reduction operations and a like number of P point 1D DFT.

- 4D $\longrightarrow$ 3D , Power of Prime Case.
  $N_1 \times N_2 \times P^n \times P^n$ 4D DFT computed by $N_1 \times N_2 \times P^n$ 3D DFT, where P is any prime (including 2) and N_1, N_2 are any numbers.

  1. Covering cubes

$$\begin{aligned} \mathcal{H}_0 &= H_3((0,0,1,m)), \quad m = 0,1,\ldots,P^n-1 \\ \mathcal{H}_1 &= H_3((0,0,rP,1)), \quad r = 0,1,\ldots,P^{n-1}-1. \end{aligned}$$

  2. Reduction stage

$$\begin{aligned} a^{\mathcal{H}_0}_{(d_1,d_2,d_3)} &= \sum_{i=0}^{P^n-1} x(d_1,\, d_2,\, d_3 - im,\, i), \qquad m = 0,1,\ldots,P^n-1 \\ a^{\mathcal{H}_1}_{(d_1,d_2,d_3)} &= \sum_{i=0}^{P^n-1} x(d_1,\, d_2,\, i,\, d_3 - rPi), \qquad r = 0,1,\ldots,P^{n-1}-1 \end{aligned}$$

$$\begin{aligned} d_1 &= 0,1,\ldots,N_1-1 \\ d_2 &= 0,1,\ldots,N_2-1 \\ d_3 &= 0,1,\ldots,P^n-1 \end{aligned}$$

  3. $N_1 \times N_2 \times P^n$ 3D DFT stage

$$\begin{aligned} V(s_1,\, s_2,\, s_3,\, s_3 m) &= \sum_{d_1=0}^{N_1-1}\sum_{d_2=0}^{N_2-1}\sum_{d_3=0}^{P^n-1} a^{\mathcal{H}_0}_{(d_1,d_2,d_3)} \omega_{N_1}^{d_1 s_1} \omega_{N_2}^{d_2 s_2} \omega_{P^n}^{d_3 s_3} \\ V(s_1,\, s_2,\, rPs_3,\, s_3) &= \sum_{d_1=0}^{N_1-1}\sum_{d_2=0}^{N_2-1}\sum_{d_3=0}^{P^n-1} a^{\mathcal{H}_1}_{(d_1,d_2,d_3)} \omega_{N_1}^{d_1 s_1} \omega_{N_2}^{d_2 s_2} \omega_{P^n}^{d_3 s_3} \end{aligned}$$

$$\begin{aligned} s_1 &= 0,1,\ldots,N_1-1, \\ s_2 &= 0,1,\ldots,N_2-1, \\ s_3 &= 0,1,\ldots,P^n-1 \end{aligned} \qquad \begin{aligned} m &= 0,1,\ldots,P^n-1 \\ r &= 0,1,\ldots,P^{n-1}-1 \end{aligned}$$

The computation of the $N_1 \times N_2 \times P^n \times P^n$ 4D DFT by this method requires $P^n + P^{n-1}$ reduction operations and a like number of 3D $N_1 \times N_2 \times P^n$ DFT.

To compute a 4D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P^n \times P^n \times P^n \times P^n$ DFT computed as $(P^n)^3(1 + 1/P + 1/P^2 + 1/P^3)$ reduction operations and a like number of P point 1D DFT.

- 4D $\longrightarrow$ 2D , Prime Case.
  $N \times P \times P \times P$ 4D DFT computed by $N \times P$ 2D DFT,
  where P is prime and N is any number.

  1. Covering planes

$$\begin{aligned} \mathcal{H}_0 &= H_2((0,1,m_1,m_2)), \quad m_1, m_2 = 0,1,\ldots,P-1 \\ \mathcal{H}_1 &= H_2((0,0,1,m_2)) \\ \mathcal{H}_2 &= H_2((0,0,0,1)) \end{aligned}$$

  2. Reduction stage

$$\begin{aligned} a^{\mathcal{H}_0}_{(d_1,d_2)} &= \sum_{i_1=0}^{P-1}\sum_{i_2=0}^{P-1} x(d_1,\, d_2 - m_1 i_1 - m_2 i_2,\, i_1,\, i_2), \qquad m_1, m_2 = 0,1,\ldots,P-1 \\ a^{\mathcal{H}_1}_{(d_1,d_2)} &= \sum_{i_1=0}^{P-1}\sum_{i_2=0}^{P-1} x(d_1,\, i_1,\, d_2 - m_2 i_2,\, i_2) \\ a^{\mathcal{H}_2}_{(d_1,d_2)} &= \sum_{i_1=0}^{P-1}\sum_{i_2=0}^{P-1} x(d_1,\, i_1,\, i_2,\, d_2) \end{aligned}$$

$$\begin{aligned} d_1 &= 0,1,\ldots,N-1 \\ d_2 &= 0,1,\ldots,P-1 \end{aligned}$$

  3. 2D $N \times P$ DFT stage

$$\begin{aligned} V(s_1,\, s_2,\, s_2 m_1,\, s_2 m_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{\mathcal{H}_0}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_P^{d_2 s_2} \\ V(s_1,\, 0,\, s_2,\, s_2 m_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{\mathcal{H}_1}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_P^{d_2 s_2} \\ V(s_1,\, 0,\, 0,\, s_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{\mathcal{H}_2}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_P^{d_2 s_2} \end{aligned}$$

$$\begin{aligned} s_1 &= 0,1,\ldots,N-1, \\ s_2 &= 0,1,\ldots,P \end{aligned} \qquad m_1, m_2 = 0,1,\ldots,P-1$$

The computation of the $N \times P \times P \times P$ 4D DFT by this method requires $P^2 + P + 1$ reduction operations and a like number of 2D $N \times P$ DFT. Note that [33] is a source of $O(P \log P)$ prime size FFT algorithms.

To compute a 4D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P \times P \times P \times P$ DFT computed by $P^3 + P^2 + P + 1$ reduction operations and a like number of P point 1D DFT.

- 4D $\longrightarrow$ 2D , Power of Prime Case.
  $N \times P^n \times P^n \times P^n$ MD DFT computed by $N \times P^n$ 2D DFT,
  where P is any prime (including 2) and N is any number.

  1. Covering planes

$$\begin{aligned} \mathcal{H}_0 &= H_2((0,1,m_1,m_2)), \quad m_i = 0,1,\ldots,P^n-1 \\ \mathcal{H}_1 &= H_2((0,r_1P,1,m_2)) \quad r_i = 0,1,\ldots,P^{n-1}-1 \\ \mathcal{H}_2 &= H_2((0,r_1P,r_2P,1)) \end{aligned}$$

  2. Reduction stage

$$\begin{aligned} a^{\mathcal{H}_0}_{(d_1,d_2)} &= \sum_{i_1=0}^{P^n-1}\sum_{i_2=0}^{P^n-1} x(d_1,\, d_2 - m_1i_1 - m_2i_2,\, i_1,\, i_2), \qquad m_1,m_2 = 0,1,\ldots,P^n-1 \\ a^{\mathcal{H}_1}_{(d_1,d_2)} &= \sum_{i_1=0}^{P^n-1}\sum_{i_2=0}^{P^n-1} x(d_1,\, i_1,\, d_2 - r_1Pi_1 - m_2i_2,\, i_2), \qquad r_1,r_2 = 0,1,\ldots,P^{n-1}-1 \\ a^{\mathcal{H}_2}_{(d_1,d_2)} &= \sum_{i_1=0}^{P^n-1}\sum_{i_2=0}^{P^n-1} x(d_1,\, i_1,\, i_2,\, d_2 - r_1Pi_1 - r_2Pi_2) \end{aligned}$$

$$\begin{aligned} d_1 &= 0,1,\ldots,N-1 \\ d_2 &= 0,1,\ldots,P^n-1 \end{aligned}$$

  3. 2D $N \times P^n$ DFT stage

$$\begin{aligned} V(s_1,\, s_2,\, s_2m_1,\, s_2m_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P^n-1} a^{\mathcal{H}_0}_{(d_1,d_2)}\omega_N^{d_1s_1}\omega_{P^n}^{d_2s_2} \\ V(s_1,\, r_1Ps_2,\, s_2,\, s_2m_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P^n-1} a^{\mathcal{H}_1}_{(d_1,d_2)}\omega_N^{d_1s_1}\omega_{P^n}^{d_2s_2} \\ V(s_1,\, r_1Ps_2,\, r_2Ps_2,\, s_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P^n-1} a^{\mathcal{H}_2}_{(d_1,d_2)}\omega_N^{d_1s_1}\omega_{P^n}^{d_2s_2} \end{aligned}$$

$$\begin{aligned} s_1 &= 0,1,\ldots,N-1, \\ s_2 &= 0,1,\ldots,P^n \end{aligned}$$

The computation of the $N \times P^n \times P^n \times P^n$ 4D DFT by this method requires $(P^n)^2(1 + 1/P + 1/P^2)$ reduction operations and a like number of 2D $N \times P^n$ DFT.

To compute a 4D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P^n \times P^n \times P^n \times P^n$ DFT computed by $(P^n)^3(1/P^2 + 1/P + 1)$ reduction operations and a like number of P^n point 1D DFT.

Chapter 3

An Algorithm for the Multiprocessor Computation of MD DFT

3.1 Introduction

This chapter describes a new algorithm for the parallel computation of the multidimensional discrete Fourier transform. The features of the algorithm are that it (1) eliminates the need for interprocessor communication and (2) that it maps to machines with any number of processors. The cost of eliminating the interprocessor communication typical for such algorithms is that one addition must be performed at each processing element for each word broadcast to the machine. The algorithm depends on a machine model that supports the communication functions broadcast and report.

The methodology proposed in this chapter is based on the $k-$dimensional hyperplane algorithm previously reviewed. The central feature of that algorithm with respect to parallel processing is that it does not intersperse communication stages with processing stages. Two basic applications of the hyperplane algorithm are considered.

The first is a direct mapping of the hyperplane algorithm to the multiprocessor. It is shown that such a mapping can be made when the number of processors is equal to the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array ($k < d$). For this case the $d-$dimensional DFT is required to be of equal and prime, or power of prime, size in $d - k + 1$ (or more) dimensions. The degree of parallelism of the algorithm is the number of $k-$dimensional hyperplanes in a covering set. The granularity of the algorithm is a single $k-$dimensional DFT.

The major benefit of the direct mapping of the hyperplane algorithm is that it eliminates the requirement for interprocessor communication for efficient MD DFT computation. The major limitation of this method are the restrictions it imposes on the machine size. There are only $d - 1$ possible mappings of the algorithm, and the degree of parallelism changes exponentially between mappings.

This limitation is addressed by an alternative mapping of the hyperplane algorithm. The variant developed is for those cases where the degree of parallelism of the machine is not matched by the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array. For these cases, the multidimensional Cooley-Tukey (MD CT) algorithm is employed together with the hyperplane algorithm. The role of the MD CT algorithm is to factor the $d-$dimensional DFT into stages of lower order $d-$dimensional

DFT. The hybrid algorithm that results has a degree of parallelism equal to the number of $k-$dimensional hyperplanes required to cover one of the resulting lower order arrays. This hybrid method allows great flexibility in matching the degree of parallelism of the algorithm to the size of the target processor.

The remainder of this chapter is organized as follows: First, a machine model is presented for a generic broadcast mode multiprocessor. Then the direct mapping of the hyperplane algorithm to the multiprocessor machine model is stated. Detailed cases of the algorithm are presented for the 2D prime, 2D power of prime (including 2), and 3D prime cases. The hybrid algorithm is introduced in chapter 4.

Machine Model

The algorithm is defined with respect to a broadcast mode multiprocessor machine model. The machine is taken to be a collection of processing elements (PEs) together with an interconnection network for interprocessor communication. The machine is externally connected to a host by a single I/O channel. All data enters the machine through the I/O channel. The communication functions that are supported must include broadcast and report.

1. *Broadcast*: This function downloads data from the I/O channel to all processing elements.

2. *Report*: This function allows a distinguished PE to upload data to the I/O channel.

Comparative Measures

Throughout, a broadcast mode multiprocessor mapping of the row-column algorithm is used for comparative purposes. That algorithm is shown in figure 3.1.

By the algorithm of figure 3.1, a 2D DFT task is seen to require alternating stages of communication and computation. The communication stages include the download of input data, the upload of transformed data, and the exchange of intermediate results between processors. The computation stages are both 1D $N-$point DFT.

3.2 Algorithm Definition

The mapping of the $k-$dimensional hyperplane algorithm of chapter 2 to the broadcast mode multiprocessor described above is given in this section. The hyperplane algorithm is computed by a reduction stage followed by a stage of $k-$dimensional DFT. The prime case of the algorithm requires

$$M = \frac{P^{d-k+1} - 1}{P - 1}$$

1. Broadcast the N rows of x uniformly among the PEs; each PE is assigned its own set of rows.
2. In parallel, compute the N independent 1D FFT(N) associated with the rows.
3. Globally transpose the intermediate results in the machine.
4. In parallel, compute the N independent 1D FFT(N) associated with the columns.
5. Upload results. The transformed data is stored column-wise in the machine. A transpose must be achieved at some point of the upload process if data is to be returned in row major form.

Figure 3.1: A multiprocessor implementation of the row-column algorithm.

reduction operations $a_{\underline{d}}^{\mathcal{H}_i}$ given by expression (2.20) of section 2.2. The power of prime case has more redundancy, and requires

$$M = \left(\frac{P^n}{P}\right)^{d-k} \cdot \left(\frac{P^{d-k+1}-1}{P-1}\right)$$

reduction operations $a_{\underline{d}}^{\mathcal{H}_i}$ given by expression (2.34) of section 2.3. Throughout the sequel, let M denote the number of reduction operations of the algorithm (*ie.* number of covering $k-$dimensional hyperplanes). For both cases the $k-$dimensional DFT are computed on data derived from the input using only additions. They produce data on hyperplanes $\mathcal{H}_i$ of the output array. These hyperplanes are defined by theorems 2.2.1 and 2.3.1 for prime and power of prime cases respectively. Conditions for implementing the hyperplane algorithm on a multiprocessor with no interprocessor communication are given by the following theorem and proof.

Theorem 3.2.1 *The optimal degree of parallelism for minimizing interprocessor communication is the number M of $k-$dimensional hyperplanes required to cover the output array.*

Proof. A k-dimensional discrete Fourier transform is performed on the $k-$dimensional array produced by each reduction operation. Hence, no interprocessor communication is required from input to output if the granularity of the reductions $a_{\underline{d}}^{\mathcal{H}_i}$ is no finer than that to put all $\underline{d} \in Z/N_1 \times \cdots \times Z/N_k$ points of an $a_{\underline{d}}^{\mathcal{H}_i}$ in a single processor. Any finer

degree would necessitate interprocessor communication between the stage in which the summations of the reduction are computed and the stage where the $k-$dimensional discrete Fourier transforms are computed.
□

Consider also that the reduction stage has no data interdependencies and can be computed simultaneously with the data download. Exploiting this concurrency places a lower limit on the partitioning of the computation. Parallel machines externally connected to an I/O channel have completion times bound by the N^2 word download and N^2 word upload. Regardless of the speed of addition, or the number of PEs, the download time of N^2 words limits the reduction stage completion time. Assuming granularity no finer than a complete reduction operation, an L PE machine reaches the limit if add time is $\lceil M/L \rceil$ faster than communication time.

All cases of the multiprocessor hyperplane algorithm are defined by the same basic structure. That structure is outlined in figure 3.2 below.

1. Assign the reduction operations $a_{\underline{d}}^{H}$ uniformly among the processing elements (PE).

2. During the download (broadcast) of the input data to the machine, each PE forms the $a_{\underline{d}}^{\mathcal{H}_i}$ terms assigned to it. For every $a_{\underline{d}}^{\mathcal{H}_i}$ term assigned to a PE only 1 addition is performed for each word broadcast to the PE. When the input download is complete, the reduction stage of the computation is also complete.

3. Each PE performs a single $k-$dimensional DFT to finish the computation.

4. Upload results and remove redundant data.

Figure 3.2: Structure of the multiprocessor hyperplane algorithm

Using the hyperplane algorithm, a MD DFT task requires stages of communication and stages of computation. The communication stages include only the data download and data upload operations. The computation stages are the DFT and data reduction operations. The communication and computation stages are the same as the serial and parallel segments of the algorithm. That is, the serial portions of the algorithm are the data download and the data upload, and the parallel segments of the algorithm are the DFT and reduction operations. Unlike the computation of the DFT, the reduction operations contain no data interdependencies, and can be computed in parallel with the download operation. On a machine with M processors, the computational speedup of the algorithm relative to a single processor computing the row-column algorithm is

$$Speedup = \frac{d}{k}(N)^{d-k}.$$

This assumes the additions of the reduction stage are computed simultaneously with the data download and incur no time penalty of their own. In the remainder of this section the algorithm is stated in some detail for the 2D prime, 2D power of prime, and the 3D prime cases.

Prime Case

The 2D prime case of the algorithm is developed below. For this case the hyperplane algorithm reduces to the line algorithm. In section 1.3, the prime case of the line algorithm was shown to be evaluated by the following two step procedure.

1. Reduction stage. Compute the $P+1$ summations:

$$a_d^{(m,1)} = \sum_{i=0}^{P-1} x(i, d-mi)$$

$$a_d^{(1,0)} = \sum_{i=0}^{P-1} x(d,i)$$

2. DFT stage. Compute the $P+1$ one-dimensional $P-$point DFT:

$$V(mt,t) = \sum_{d=0}^{P-1} \omega_P^{dt} a_d^{(m,1)}, \qquad m = 0,1,\ldots,P-1$$

$$V(t,0) = \sum_{d=0}^{P-1} \omega_P^{dt} a_d^{(1,0)}, \qquad t = 0,1,\ldots,P-1$$

The summation terms $a_d^{(m,1)}$ reduce the input data to the vectors that must be transformed in order to obtain the output along the lines $L((m,1))$. The summation term $a_d^{(1,0)}$ gives the vector that must be transformed to obtain the output along the line $L((1,0))$. The computation of the $P \times P$ line algorithm on a broadcast mode multiprocessor is realized by the procedure of figure 3.3.

1. Assign the computation of the reduction operations $a^{(x,y)}$ evenly among M or fewer PEs.

2. Broadcast the rows of input data to the PEs and simultaneously compute the reductions $a^{(x,y)}$ at the PEs. The reduction operations are computed as follows: When row i is received, the PE assigned $a^{(1,0)}$ sums the elements of the row and places that sum in position i of $a^{(1,0)}$. In a similar manner, the PE assigned $a^{(m,1)}$ rotates row i mi positions to the right and sums it componentwise to the other received rows. In this fashion $a_d^{(m,1)}$ will exist at position d of the first row received. The rotation can be achieved by an address offset and modulo P address arithmetic.

   Note that the partial result due to row i is accumulated as row i is received. In this way each PE requires only $O(N)$ storage for each $a^{(x,y)}$ it is assigned.

3. In parallel, each PE computes a $P-$point 1D DFT for every reduction $a^{(x,y)}$ assigned to it.

4. Upload data to host. There is some redundancy in the data, and the data is permuted among the PEs.

   Redundancy: For this case, the redundancy is trivially that output point $V(0,0)$ is common to every PE.

   Permutation: The permutation is that every PE contain a line of output data as specified by theorem 1.3.1. The row 0 and column 0 are in the PEs which computed the $a^{(0,1)}$ and $a^{(1,0)}$ terms, and are not permuted. Element l of the vector produced by the PE that computed $a^{(m,1)}$ is the column l and row $(N - ml) \bmod P$ element of the 2D DFT.

Figure 3.3: 2D $P \times P$ multiprocessor mapping of the line algorithm.

To obtain a measure of the performance of the algorithm, allow a comparison of a multiprocessor with $P+1$ PEs running the parallelized line algorithm to a single processor computing the row column algorithm. Let the computational burden of the reduction stage be taken simultaneously with the data download. Each reduction operation for the 2D $P \times P$ case requires $O(P^2)$ additions. Then the speedup of the parallel line algorithm on $P+1$ processors relative to the row-column method on a single processor is

$$Speedup = \frac{2 \cdot P \cdot FFT(P)}{FFT(P)} = 2P.$$

The algorithm achieves a $2P$ speedup with $P+1$ processors rather than the expected P speedup because the number of 1D DFT has been reduced from $2P$ to $P+1$. The justification for not including the time cost of the additions of the reduction stage is that these additions are concurrent with the data download operation, which is common to both algorithms and can not be parallelized.

The performance improvement of the algorithm depends entirely on the computation of the reduction operations in parallel with the data download. Consider that a single processor computing just the reduction stage of the line algorithm would require $O(P^3)$ additions. Below, the mapping of the 2D power of prime and 3D prime cases of the algorithm are considered.

Power of Prime Case

This section gives the power of prime case of the algorithm specialized for $2^n \times 2^n$ 2D DFT. The main differences between the prime and power of prime cases are the computation of the reduction stage, and the removal of redundant data. The power of prime case of the line algorithm is given below.

1. Reduction stage. Compute the $2^n + 2^{n-1}$ summations:

$$a_d^{(m,1)} = \sum_{i=0}^{2^n-1} x(i, d-mi)$$

$$a_d^{(1,2s)} = \sum_{i=0}^{2^n-1} x(d-2si, i)$$

2. DFT stage. Compute $2^n + 2^{n-1}$ one-dimensional 2^n−point DFT:

$$V(mt,t) = \sum_{d=0}^{2^n-1} \omega^{dt} a_d^{(m,1)}, \quad m = 0,1,\ldots,2^n-1$$

$$V(t,2st) = \sum_{d=0}^{2^n-1} \omega^{dt} a_d^{(1,0)}, \quad s = 0,1,\ldots,2^{n-1}-1$$

$$t = 0,1,\ldots,2^n-1$$

The computation of the $a^{(m,1)}$ terms is the same as the prime case. That is, when row i is received, the PE assigned $a^{(m,1)}$ rotates it mi positions to the right, and adds component-wise to the previously received rows. This compresses the data so that the storage requirement for this stage is only that of a single row.

The computation of the $a^{(1,2s)}$ terms is similar. The PE assigned $a^{(1,2s)}$ rotates column i $2si$ positions to the right, and adds component-wise to the other columns. The problem this introduces is that the data is downloaded in row major order. The following procedure computes the $a^{(1,2s)}$ terms given row-major data.

Let α be the maximum such that $2^\alpha | 2s$, and let $x(i,j)$ denote element j of row i. As row i is received, the PE sums all elements of the row which are $2^{n-\alpha}$ positions apart. There will be $2^{n-\alpha}$ such sums each of 2^α elements. These can be expressed as

$$c_{i,l} = \sum_{k=0}^{2^\alpha - 1} x(i,\ l + 2^{n-\alpha}k), \quad l = 0, 1, \ldots, 2^{n-\alpha} - 1.$$

The l^{th} of these $2^{n-\alpha}$ sums of the elements of row i are accumulated with the $(i + 2sl) \bmod 2^n$ element of $a^{(1,2s)}$. Essentially, the $c_{i,l}$ terms of any $a^{(1,2s)}$ are computed by striding through a received row by $2^{n-\alpha}$ and forming an accumulated sum of those points. The following example illustrates this procedure.

Example: Consider a 4×4 2D DFT. The reduction operations $a^{(1,2s)}$ are required for $s = 0, 1$. Using the procedure outlined above, the reductions are formed from downloaded rows in the following manner.

For $s = 0$, the maximum α such that $2^\alpha | 0$ is n. The PE assigned this term must add elements of each row that are 1 apart; all elements of each row received are summed together. The term $a_d^{(1,0)}$ is the accumulated sum of row d.

For $s = 1$, the required reduction operation is $a^{(1,2s)}$. The maximum α such that $2^\alpha | 2s$ is $\alpha = 1$. As row i is received, the PE sums elements which are 2 positions apart. There will be 2 such sets formed for each row. These are:

$$\begin{aligned} c_{0,0} &= x_{0,0} + x_{0,2} & \qquad c_{0,1} &= x_{0,1} + x_{0,3} \\ c_{1,0} &= x_{1,0} + x_{1,2} & \qquad c_{1,1} &= x_{1,1} + x_{1,3} \\ c_{2,0} &= x_{2,0} + x_{2,2} & \qquad c_{2,1} &= x_{2,1} + x_{2,3} \\ c_{3,0} &= x_{3,0} + x_{3,2} & \qquad c_{3,1} &= x_{3,1} + x_{3,3}. \end{aligned}$$

After row i is received and the $c_{i,l}$ are formed, the elements of $a_d^{(1,2)}$ are given as:

$$\begin{aligned} a_0^{(1,2)} &= c_{0,0} + c_{2,1} \\ a_1^{(1,2)} &= c_{1,0} + c_{3,1} \\ a_2^{(1,2)} &= c_{2,0} + c_{0,1} \\ a_3^{(1,2)} &= c_{3,1} + c_{1,1}. \end{aligned}$$

This finishes the reduction stage of the algorithm. The 1D DFT are performed on points

$$a_d^{(x,y)}, \quad d = 0, \ldots, P-1$$

to complete the computation.
□

As in the prime case, when the computation is complete there are some output points that appear in more than one processor. The procedure of figure 3.4 allows the data to be uploaded with the redundant points removed. In the procedure, $L((x,y))$ denotes the line of output computed by the PE that was assigned the reduction operation $a^{(x,y)}$.

---

1. Upload all points of the line $L((0,1))$.
2. For each $\alpha = 0, 1, \ldots, n-1$, define $j = 2^\alpha, \ldots, 2^{\alpha+1} - 1$ and upload all points on the lines $L((j,1))$ except multiples of $2^{n-\alpha}$.
3. Upload all points of the line $L((1,0))$ except $(0,0)$.
4. For each $\alpha = 0, 1, \ldots, n-2$, define $k = 2^\alpha, \ldots, 2^{\alpha+1} - 1$ and upload all points on the lines $L((1,2k))$ except multiples of $2^{n-\alpha-1}$.

Figure 3.4: $2^n \times 2^n$ data upload procedure.

Example: Consider a 4×4 2D DFT. The array below shows the effect of the upload strategy of figure 3.4 on the set of covering lines. The leftmost column of the array enumerates the lines $L((x,y))$ that cover the output array. All elements of the vector $L((x,y))$ are listed in the corresponding row of the array. The elements that are not

uploaded are boxed.

$$
\begin{array}{llllll}
L((0,1)) & V_{0,0} & V_{0,1} & V_{0,2} & V_{0,3} \\
L((1,1)) & \boxed{V_{0,0}} & V_{1,1} & V_{2,2} & V_{3,3} \\
L((2,1)) & \boxed{V_{0,0}} & V_{2,1} & \boxed{V_{0,2}} & V_{2,3} \\
L((3,1)) & \boxed{V_{0,0}} & V_{3,1} & \boxed{V_{2,2}} & V_{1,3} \\
L((1,0)) & \boxed{V_{0,0}} & V_{1,0} & V_{2,0} & V_{3,0} \\
L((2,0)) & \boxed{V_{0,0}} & V_{1,2} & \boxed{V_{2,2}} & V_{3,2}
\end{array}
$$

Observe that all the elements of any vector (line) not uploaded from a PE are multiples of 2 apart.
□

Note that line algorithm required 6 1D DFT to compute the DFT of the example. This compares to the 8 1D DFT required by the row-column algorithm. Given that the additions of the reduction operations are performed concurrently with the input data download, a 6 PE machine could compute this DFT in the time required for the data upload and download, plus the computation time for one 1D DFT. This represents a speedup in the DFT computation of 8 relative to a single processor executing the row-column method.

If the allowed granularity is no smaller than a 1D DFT the maximum degree of parallelism of the row-column method is only 4, even though 8 1D DFT are required. This is due to the fact that the 1D DFT of the second stage (*ie.* column) are dependent on the first stage. On a 4 PE machine, the row-column method would require time for the data upload and download, time for 2 1D DFT, and time for a global data transpose that requires every PE to exchange data with every other PE. If we assume that the global transpose requires zero time, then the speedup of this method relative to a single processor computing the row-column method is 4.

The parallelized row-column method for $N \times N$ 2D DFT, $N = 2^n$, gives a linear speedup limited by N (disregarding transpose time). The parallel line algorithm attains speedup of $2N$ given $3N/2$ PEs. This "super linear" performance is due to the reduction of the number of 1D DFT from $2N$ to $3N/2$ and to the fact that the additions of the reduction stage were counted with the communication operations of the data download. The major limitation of this method is that it does not scale to machines whose degree of parallelism is lower than $3N/2$. This constraint will be eliminated in later sections. Before proceeding to that work the 3D prime case of the algorithm is discussed.

3D Case

The general mapping of the 3D prime case of the hyperplane algorithm is presented below. The primary motivation for the algorithm is that it exchanges increased computation time (larger granularity) for a decreased degree of parallelism relative to the parallel line method (k=1).

The example covered in this section is the 3D $P \times P \times P$ DFT, where P is a prime number. If the line algorithm were applied (case of $k = 1$), a number of lines equal to $P^2 + P + 1$ would be required to cover the output array, and therefore a like number of processors is needed. To reduce the degree of parallelism of the computation, take $k = 2$ and compute planes of the output. For this case the degree of parallelism is $P+1$ (number of planes in a covering set) and the granularity is a single $P \times P$ 2D DFT.

In the previous section the convention that data was input to the machine in row-major order was adopted. In order to remain consistent with this convention it is necessary to consider a different set of covering planes that those given in theorem 2.2.1. Allow the covering planes to be given by

$$\begin{aligned} H_2((0,m,1)) &= \{(s_1,\, s_2 m,\, s_2) : s_1, s_2 \in Z/P\}, \\ H_2((0,1,0)) &= \{(s_1,\, s_2,\, 0) : s_1, s_2 \in Z/P\}, \\ & m = 0, 1, \ldots, P-1. \end{aligned} \tag{3.1}$$

Various data flows can be obtained by permuting the order of the components of the hyperplanes of theorems 2.2.1 and 2.3.1. In order for the sets to maintain their covering property each hyperplane must be permuted in the same way.

For the hyperplanes of (3.1) the algorithm is given by the following procedure:

1. Reduction stage. Compute the $P + 1$ summations:

$$\begin{aligned} a_{d_1,d_2}^{H((0,m,1))} &= \sum_{i=0}^{P-1} x(d_1, i, d_2 - mi) \\ a_{d_1,d_2}^{H((0,1,0))} &= \sum_{i=0}^{P-1} x(d_1, d_2, i) \end{aligned}$$

2. DFT stage. Compute the $P + 1$ 2D DFT:

$$\begin{aligned} V(s_1, s_2 m, s_2) &= \sum_{d_1=0}^{P-1} \sum_{d_2=0}^{P-1} a_{d_1,d_2}^{H((0,m,1))} \omega^{s_1 d_1} \omega^{s_2 d_2}, \\ V(s_1, s_2, 0) &= \sum_{d_1=0}^{P-1} \sum_{d_2=0}^{P-1} a_{d_1,d_2}^{H((0,1,0))} \omega^{s_1 d_1} \omega^{s_2 d_2}, \end{aligned}$$

$$m = 0, 1, \ldots, P-1 \quad \text{and} \quad s_1, s_2 = 0, 1, \ldots, P-1.$$

The computation of the 3D prime algorithm on a multiprocessor is essentially the same as the 2D case given in figure 3.3. They key differences between them are the evaluation of the reduction operations and the removal of the redundancy in the data upload. These are specified below.

Assign the $P+1$ reduction operations a^H to the PEs. During the download operation the rows of input data are broadcast to the machine. Let row (r_1, r_2) denote the P input points $x_{(r_1,r_2,0)}, \ldots, x_{(r_1,r_2,p-1)}$ of the input array. The PE assigned the computation of

$$a_{d_1,d_2}^{H((0,1,0))}$$

forms an accumulated sum of the elements of row (r_1, r_2) when it is received. That sum is placed in location (r_1, r_2) of $a^{H((0,1,0))}$. The computation of this term requires one addition for each word input to the PE, and is completed when the last input enters the PE.

The PEs assigned the computation of the $a^{H((0,m,1))}$ terms compute as follows: When row (r_1, r_2) of the input is received it is rotated mr_2 positions to the right and summed componentwise with the elements of row r_1 of $a^{H((0,m,1))}$. The rotation is achieved by address offset and modulo N address arithmetic. Similar to the 1D case, the summation operation compresses the data. Therefore the reduction stage requires the PE to have storage for only P^2 points for each reduction operation assigned to it.

After the reduction stage is finished, the 2D DFT are performed on the a^H in each PE. *This completes the computation.* The output is distributed among the PEs as hyperplanes. There are output points that were computed in more than one PE. It is desirable that these be removed before the results are uploaded. The redundant points are all on the the line $L((1,0,0))$ of the output array. In order to eliminate these points during the data upload, only one PE is required to upload the points on the line $L((1,0))$ in its 2D output array. That is, only one PE uploads the transformed points $a_{0,0}^H, a_{1,0}^H, \ldots, a_{P-1,0}^H$.

For the 3D prime case the multiprocessor hyperplane algorithm requires a degree of parallelism equal to $P+1$ and the granularity of a single 2D $P \times P$ DFT. Therefore, the computational speedup relative to a single processor computing the row-column method is

$$Speedup = \frac{3 \cdot P^2 \cdot FFT(P)}{2 \cdot P \cdot FFT(P)} = \frac{3P}{2}.$$

This does not account for the data upload and download or the additions of the reduction operations that are performed simultaneously with the download.

For comparative purposes, consider a P processor machine computing the row-column like procedure of figure 3.5. The modified row-column algorithm of figure 3.5 requires communication and computation stages. The communication stages include the upload and download of data points, and the exchange on intermediate results (global transpose). The computation stages are 1D and 2D DFT. If the 2D DFT at each PE are computed by the row-column method, then this algorithm is seen to

1. Broadcast two dimensions of data to each PE.
2. Compute 2D $P \times P$ DFT at each PE.
3. Globally transpose the data in the machine.
4. Compute 1D DFT on the remaining dimension.
5. Upload the data.

Figure 3.5: A modified row-column 3D algorithm.

require the time to compute $3P$ 1D DFT. Its computational speedup relative to a single processor computing the row-column method is

$$Speedup = \frac{3 \cdot P^2 \cdot FFT(P)}{3 \cdot P \cdot FFT(P)} = P.$$

The P processor implementation of the row-column like procedure of figure 3.5 has a speedup of P, but the $P+1$ processor implementation of the hyperplane algorithm has a speedup of $3P/2$. The hyperplane algorithm achieves a 50% improvement over the row-column method at the cost of only a single processor. The advantage of the hyperplane method is due to its overall reduced number of DFT relative to the row-column method.

Conclusion

This section demonstrated a multiprocessor algorithm for MD DFT computation. The algorithm is based on a direct mapping of the hyperplane algorithm to the multiprocessor architecture. The algorithm is shown to naturally partition the $d-$dimensional DFT into M independent computations, M equal to the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array.

For multiprocessor architectures a mapping is given that requires no interprocessor communication, and allows the M independent computations to occur concurrently with the input download. On single I/O channel machines that are capable of exploiting the full degree of parallelism of the algorithm, execution times as low as the time to compute a single one dimensional FFT on N points, plus the time to upload and download the data are attainable. If task level pipelining is used, average times equal to the single channel I/O limit occur, but single task completion times are longer.

Furthermore, if the input data is real, the only stage of the algorithm that inputs complex data is the data upload.

The primary benefit of the algorithm was seen to be that it requires no interprocessor communication. The primary limitation of the algorithm is that it does not scale flexibly to the degree of parallelism of the target processor. This issue is addressed in the next section.

Chapter 4

A Hybrid Algorithm for the Multiprocessor Computation of MD DFT

This chapter combines the hyperplane and multidimensional Cooley-Tukey algorithms to produce a highly scalable broadcast mode multiprocessor algorithm. The main features of the algorithm are that it requires no interprocessor communication, and that it maps to machines of varying degrees of parallelism. The cost of eliminating interprocessor communication by this method is that one addition must be performed at each processing element for each input broadcast to the machine.

Overview

In previous sections the $k-$dimensional hyperplane algorithm was mapped to a broadcast mode multiprocessor machine model. The parallel algorithm that resulted requires no interprocessor communication. However, in order to exploit that benefit, the number of processors in the machine has to equal the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional output array. For many cases this can be a large number. Consider that the power of prime case (including 2) of the algorithm requires the degree of parallelism of the machine to be

$$\left(\frac{P^n}{P}\right)^{d-k}\left(\frac{P^{d-k+1}-1}{P-1}\right).$$

The granularity of the corresponding computation is a single
$k-$dimensional DFT of size $(P^n)^k$.

For a given problem size the only means available for adjusting the degree of parallelism is to alter k. The effect this has on the algorithm is to change the dimension of the hyperplanes used to cover the output array. An increase in k increases the granularity of the computation performed at each PE by increasing the dimension of the required DFTs. This increase in granularity is accompanied by an associated decrease in the degree of parallelism required by the algorithm.

Using this method, the tradeoff between granularity and degree of parallelism is limited to powers of P^n. That is, for an increase in dimension of 1, the size of the computation increases P^n times and the degree of parallelism decreases roughly P^n

times. At the limit $k = d - 1$, this method allows a minimum degree of parallelism of

$$(P^n)\left(\frac{P+1}{P}\right).$$

The corresponding granularity of the computation performed at each PE is a $(P^n)^{d-1}$ DFT. For the $N = 2^n$ case, the minimum degree of parallelism is $3N/2$, and the corresponding granularity is a single $d-1$ dimension DFT of size $(N)^{d-1}$.

The primary limitations that result from manipulating only the dimensionality of the problem are: (1) that the degree of parallelism can only be scaled by powers of P^n, and (2) that there are only $d-1$ possible mappings of the algorithm.

One method for obtaining increased control over the degree of parallelism of the algorithm is to modify the order (size) of the computation in each dimension. This can be done using multidimensional Cooley-Tukey (MD CT) / vector-radix techniques. The feature of the MD CT algorithms central to this work is that they can factor a $d-$dimensional DFT into stages of lower order $d-$dimensional DFT.

In the next section an algorithm that marries the hyperplane algorithm to the MD CT algorithm is introduced. The resulting hybrid algorithm uses the MD CT algorithm to reduce the order of the computations performed in each dimension, and the hyperplane algorithm to reduce the number of dimensions. The hybrid algorithm requires no interprocessor communication, and can be scaled to match the degree of parallelism of the target machine.

4.1 Derivation

The hybrid algorithm will be developed as follows: First the structure of the MD CT factorization is reviewed. Then a nesting of the hyperplane algorithm within the MD CT factorization is given that permits the MD DFT to be computed with a reduced degree of parallelism and no interprocessor communication.

MD CT Components

The structure of the MD CT (vector-radix) factorization of the 2D $n \times n$ DFT where $n = rs$ is given by the block diagram in figure 4.1. The diagram describes the relationship between the first and second DFT stages of a MD CT factorization for $n = rs$. In the diagram, each DFT of a stage is identified by a rectangle with an index (x, y) affixed to it. DFT i of stage two refers to the DFT whose index (x, y) is the i^{th} taken lexicographically. The diagram shows graphically that input j to DFT i of stage two is from output i of DFT j of stage one.

The computation and data flow of 4.1 are defined in terms of the decomposition of the input and output index sets. The procedure below defines the computation in terms of the coset decomposition of the index set.

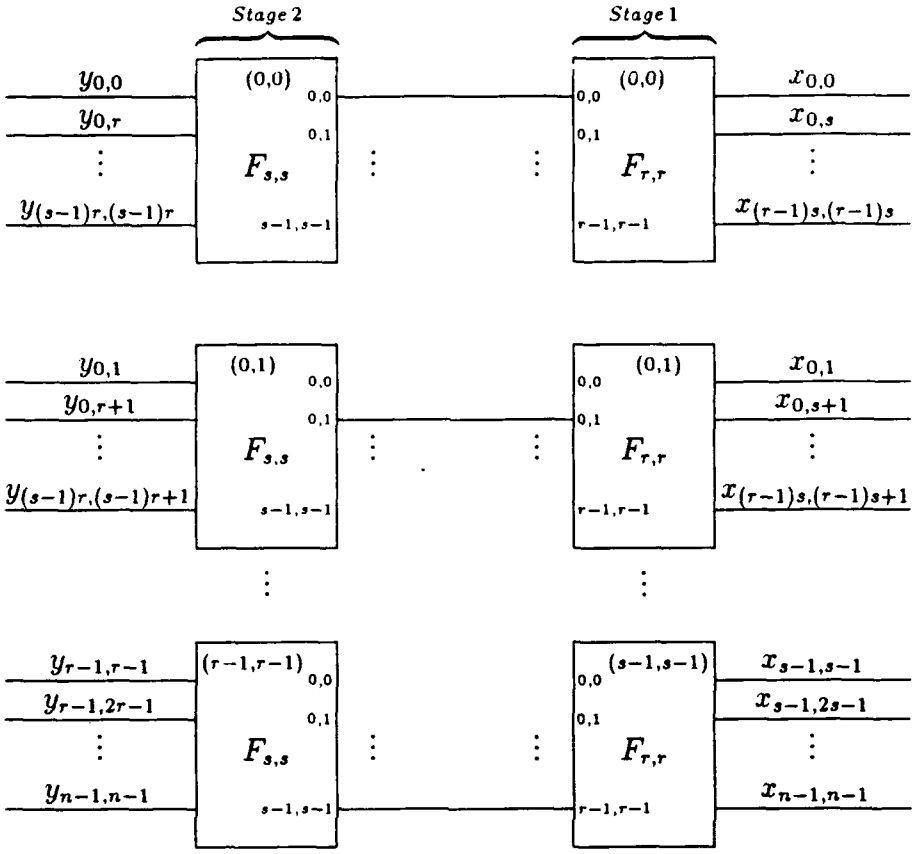


Figure 4.1: Flow diagram of an $n \times n$ 2D FFT factorization for $n = rs$.

1. Computation of $d-$dimensional DFTs $F_{\underline{r}}$. Each of these is performed on a coset of the input with respect to the subgroup $\underline{s}Z/\underline{n}$.

2. Twiddle multiplications. The element $\underline{a}$ of the input coset $\underline{u}$ that was transformed in step (1) is multiplied by $\omega_{n_1}^{a_1 u_1} \cdots \omega_{n_d}^{a_d u_d}$.

3. Computation of $d-$dimensional DFTs $F_{\underline{s}}$. Each of these produces a coset of output with respect to the subgroup $\underline{r}Z/\underline{n}$. The input and output subgroups are dual.

This procedure relates directly to the diagram of figure 4.1. The DFTs of stage one of the figure correspond to the $F_{\underline{r}}$ of step one of the procedure, and the DFTs of stage two of the figure correspond to step three. Step two of the procedure defines the twiddle multiplications. The diagram shows the input points to each DFT of stage one. The DFT labeled $(0,0)$ is seen to be computed on the points whose indexes are given by the set

$$\{(x,y) : (x,y) \in sZ/n \times sZ/n\}.$$

Similarly, the DFT labeled $(0,1)$ is computed on the points whose indexes are in the set

$$\{(x,y)+(0,1) : (x,y) \in sZ/n \times sZ/n\}.$$

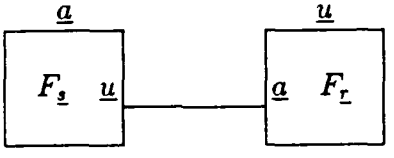
That is, the input data for the DFT labeled $(0,0)$ are the points whose indexes are the subgroup $\underline{s}Z/\underline{n}$ of $Z/\underline{n}$, and the input data to the DFT labeled $(0,1)$ are the points whose indexes are the $(0,1)$ coset of the decomposition of $Z/\underline{n}$ with respect to the subgroup $\underline{s}Z/\underline{n}$. In general the input to the DFT labeled (x,y) is coset (x,y) of the decomposition. The cosets of the decomposition of $Z/\underline{n}$ with respect to $\underline{s}Z/\underline{n}$ can be enumerated by $\underline{u} \in Z/s \times Z/s$. In this way $F_{\underline{r}}$ number $\underline{u}$ is said to transform coset $\underline{u}$ of the input array.

The second stage of DFT can be described in a similar way. The difference between the stages is that the DFTs of stage two each produce a coset of the output array with respect to the subgroup $rZ/n \times rZ/n$. The cosets in this decomposition are enumerated by $\underline{a} \in Z/r \times Z/r$. In this way $F_{\underline{s}}$ number $\underline{a}$ produces the coset $\underline{a}$ of the output array. These are the output points whose indexes are given by

$$\{(x,y)+(a_1,a_2) : (x,y) \in rZ/n \times rZ/n\}.$$

Above, the input and output data flow has been described in terms of the decomposition of the input and output index sets. Below, the data flow between DFT stages is described in terms of those same decompositions.

The output of each $F_{\underline{r}}$ of stage one is associated with the index set $\underline{a} \in Z/r \times Z/r$. These correspond to the indexes marked on the left column of the stage one DFT in the diagram. Similarly, the input to each DFT of stage two is associated with the index set $\underline{u} \in Z/s \times Z/s$. These correspond to the indexes marked on the right column of the DFT of stage two. Using these associations, the data flow between the DFT stages can be described in terms of the decompositions of the input and output index sets. Recall from the discussion above that an $F_{\underline{s}}$ of stage two is labeled by $\underline{a} \in Z/\underline{r}$, and an $F_{\underline{r}}$ is labeled by $\underline{u} \in Z/\underline{s}$. Then the $F_{\underline{s}}$ associated with coset $\underline{a}$ of the output array takes its input $\underline{u}$ from the output $\underline{a}$ of the $F_{\underline{r}}$ associated with input coset $\underline{u}$. This relation is depicted graphically below.



Evaluating the MD DFT by the factorization of (4.1) requires computing DFT stages composed of $F_{\underline{r}}$ and $F_{\underline{s}}$. The inputs to the i^{th} $F_{\underline{s}}$ were shown to be the collection of i^{th} outputs of every $F_{\underline{r}}$. Let that $F_{\underline{s}}$ be assigned to a PE in a multiprocessor. The PE can evaluate that $F_{\underline{s}}$ if the i^{th} point from every stage one $F_{\underline{r}}$ is available at the PE. Below, an assignment of the hyperplane algorithm to the DFTs of stage one is given that satisfies this criterion.

Hyperplane Components

The multiprocessor hyperplane algorithm developed in the previous sections of this chapter will be applied to the computation of the DFTs of the first stage of the MD CT factorization. Before proceeding with the development of the algorithm, first consider the computation of a single $F_{\underline{r}}$ by that method.

Assume that $F_{\underline{r}}$ is of the same size in $d-k+1$ dimensions, and let $r_k = \cdots = r_d$. Then the indexing set of $F_{\underline{r}}$ may be written

$$\begin{aligned} \mathcal{C} &= Z/\underline{r} = Z/r_1 \times \cdots \times Z/r_d = \mathcal{A} \times \mathcal{B}, \\ \mathcal{A} &= Z/r_1 \times \cdots \times Z/r_{k-1} \times Z/r_k, \\ \mathcal{B} &= (Z/r_k)^{d-k}. \end{aligned}$$

The number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array $\mathcal{C} = Z/\underline{r}$ is denoted $M(\underline{r})$. Let the number of PEs in the machine be $M(\underline{r})$, and recall that on such a machine the DFT $F_{\underline{r}}$ may be computed by the procedure listed below.

---

1. Assign one hyperplane of the covering set to each PE.

2. Broadcast the inputs to the machine, and simultaneously compute the reduction operations.

3. Compute a single $k-$dimensional DFT over $\mathcal{A}$.

Figure 4.2: Computation of $F_{\underline{r}}$ at a PE.

After this procedure is complete each PE has its own $k-$dimensional hyperplane of the outputs of the $F_{\underline{r}}$. For each of the remaining $F_{\underline{r}}$ of stage one, apply the same assignment of hyperplanes to PEs that was used for the first $F_{\underline{r}}$. These DFTs are computed in the same manner as the first.

When the computation of this stage is complete, every PE has one $k-$dimensional hyperplane from each $F_{\underline{r}}$ of stage one. Since each PE was assigned the same hyperplane in every $F_{\underline{r}}$, the PE that has point $\underline{a}$ from the first $F_{\underline{r}}$ has point $\underline{a}$ from every $F_{\underline{r}}$ of stage one. In the previous section these were shown to be the points needed to compute coset $\underline{a}$ of the output partition. The PE generates coset $\underline{a}$ of the output partition by applying the appropriate twiddle multiplications to the stage one outputs, then computing a single $d-$dimensional DFT $F_{\underline{s}}$.

Below an example is given to illustrate this procedure. The example is for a 9×9 2D DFT computed on a four PE broadcast mode multiprocessor.

Example: Consider the problem of computing a 9×9 2D DFT on a 4 processor machine. A direct mapping of the line algorithm ($k = 1$ case of the hyperplane algorithm) for this computation would require 12 processors. However, the hybrid algorithm using a 3×3 factorization requires only 4 processors. This algorithm will be developed below.

The computation depends on the evaluation of the 3×3 2D DFT $F_{3,3}$, by the line algorithm. That computation is

$$\underline{z} = F_{3,3}\,\underline{x},$$

and is described by the example at the end of section 1.3. The output array of the 3×3 2D DFT is given by four lines. These are:

$$\begin{aligned} L((0,1)) &= \{z_{0,0}\,,\ z_{0,1}\,,\ z_{0,2}\} \\ L((1,1)) &= \{z_{0,0}\,,\ z_{1,1}\,,\ z_{2,2}\} \\ L((2,1)) &= \{z_{0,0}\,,\ z_{2,1}\,,\ z_{1,2}\} \\ L((1,0)) &= \{z_{0,0}\,,\ z_{1,0}\,,\ z_{2,0}\} \end{aligned}$$

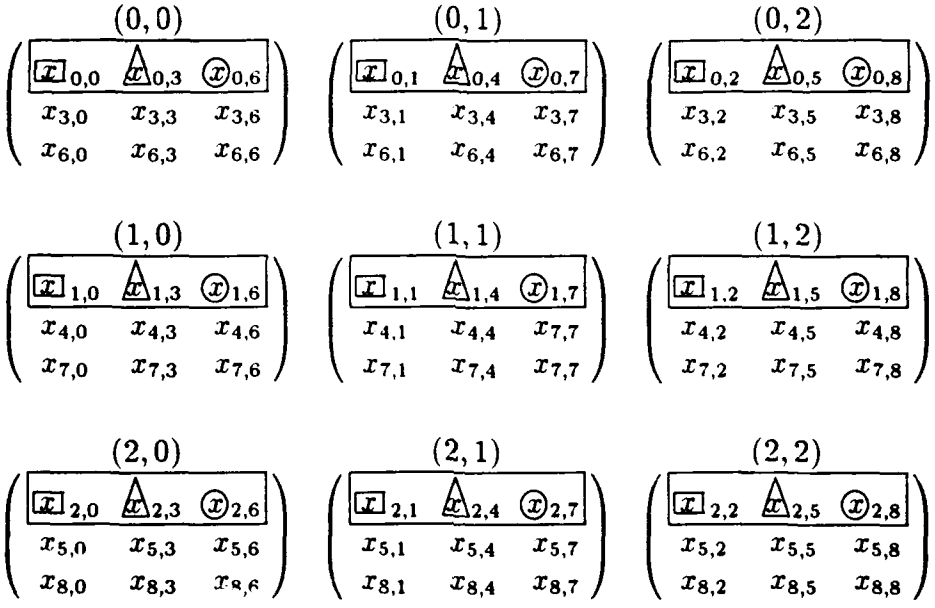
Each of these lines is assigned to a different processor. Let the output on the lines $L((m,1))$, $m = 0,1,2$ be computed by PE_m, and the output on the line $L((1,0))$ be computed by PE_3. As the inputs are broadcast to the machine, each PE computes the reduction operation $a^{(x,y)}$ associated with its assigned line of the output. After the broadcast is complete, each PE performs a 3–point DFT to produce its set of output points. Each of the 3×3 2D DFT of the first DFT stage of the factorization are computed in this way.

The 9×9 2D DFT factored into stages of 3×3 DFT is given by [14]

$$Y(a_1 + 3b_1, a_2 + 3b_2) = \overbrace{\sum_{u_1=0}^{2}\sum_{u_2=0}^{2} \omega_3^{u_1 b_1}\omega_3^{u_2 b_2}\,\omega_9^{a_1 u_1}\omega_9^{a_2 u_2}}^{Stage\ 2} \cdot \underbrace{\sum_{p_1=0}^{2}\sum_{p_2=0}^{2} x(3p_1 + u_1, 3p_2 + u_2)\omega_3^{a_1 p_1}\omega_3^{a_2 p_2}}_{Stage\ 1} \tag{4.1}$$

The first stage of DFT of this factorization requires the computation of nine 2D 3×3 DFT. The inputs to these DFT are listed in the arrays below. Each of the arrays is

labeled above by the index of its first element in parenthesis.



This partition of the input elements results from decimating both dimensions of the array simultaneously. Each array of the partition is a coset of $Z/9 \times Z/9$ with respect to the subgroup $3Z/9 \times 3Z/9$. The label above each array is its coset leader. Throughout this example, the arrays will be defined and referred to as cosets, and distinguished by their coset leaders.

Each of the nine required 3×3 2D DFT of stage one are computed by the multiprocessor line algorithm as described above. The lines are taken over each of the cosets. The assignment of lines to PEs is the same for each of the cosets. The PE assigned line $L((0,1))$ in coset $(0,0)$ is assigned line $L((0,1))$ in every coset.

For simplicity of presentation, allow the input data to be broadcast to the PEs by coset. Then the reduction operations are performed simultaneously with the data download. Each PE performs one addition for each word it inputs. As all the points of a coset are received, the associated reduction operation is completed. When the download is finished the PE performs nine independent 3−point 1D DFT to complete stage one.

Consider the computation of the line $L((0,1))$ in each coset of the input partition. For each coset downloaded the PE computes the required reduction operation. When the download is complete, the PE performs the 1D DFT. At this time the PE contains all the stage one outputs corresponding to the first row of each coset shown above (enclosed in rectangles). The appropriate twiddle multiplications are applied at this time.

The second stage of DFTs requires computing nine 3×3 2D DFT. To obtain all output points by the method described in this section, each PE must compute three 3×3 2D DFT. For the PE assigned line $L((0,1))$ of each stage one coset, one 2D DFT

is computed for each element in the line. That is, the first 2D DFT is performed on all points enclosed in squares, the second is computed on all points enclosed in triangles, and the third 2D DFT is computed on all points enclosed in circles. After this stage of 2D DFT is complete, the entire computation is distributed by output coset throughout the machine. This permutation is similar to the permutation that results from a typical application of the vector-radix algorithm.
□

The factorization of the 9×9 2D DFT given in the example above contains 18 $F_{3,3}$ 2D DFT. If each of these were evaluated by the row-column method there would be 108 FFTs of 3−points each. Assume the reduction operations of the line algorithm are performed concurrently with the input download and the communication time is not considered, then the computational speedup of the hybrid algorithm of the example relative to a single processor computing the 3×3 factorization is

$$Speedup = \frac{108\,FFT(3)}{27\,FFT(3)} = 4,$$

which is the ideal linear speedup, and requires no interprocessor communication.

The multiprocessor computation of $F_{\underline{n}}$ was described in previous sections of this chapter. The procedure that follows describes the multiprocessor computation of $F_{\underline{n}}$ factored into stages of $F_{\underline{s}}$ and $F_{\underline{r}}$.

1. Associate one k−dimensional hyperplane of $Z/\underline{r}$ with each PE. The required hyperplanes are given by theorems 2.2.1 and 2.3.1. A hyperplane represents a subset of the output of a single $F_{\underline{r}}$.

2. The input is partitioned into cosets with respect to the subgroup $\underline{s}Z/\underline{n}$. These are enumerated by $\underline{u} \in Z/\underline{s}$. Each corresponds to an array $Z/\underline{r}$. Assume for simplicity of presentation that the input has been ordered by coset.

3. Broadcast the input to the machine. As coset $\underline{u}$ is input, each PE computes the reduction operation a^H corresponding to the hyperplane assignment of step (1). For each coset, this step is identical to the multiprocessor computation of a single $F_{\underline{r}}$. When this step is complete, s independent reduction operations a^H have been computed by each PE.

4. Compute a number s of k−dimensional DFT at each PE. Each k−dimensional DFT is computed on the output of a reduction operation from step (3). Each produces the hyperplane of the coset associated with the PE in step (1).

5. Apply the twiddle multiplications.

6. Compute independent DFTs $F_{\underline{s}}$ at each PE. There are s k−dimensional hyperplanes in each PE. Each hyperplane has $|\mathcal{A}|$ points. $|\mathcal{A}|$ DFTs are performed. For $\underline{x} \in \mathcal{A}$, a DFT $F_{\underline{s}}$ is performed on the collection of all points with index $\underline{x}$ from each hyperplane in the PE. After this step the computation is complete.

7. Data upload. The output is distributed among the PEs according to the coset decomposition of $Z/\underline{n}$ with respect to the subgroup $\underline{r}Z/\underline{n}$. This is a normal vector-radix [14] permutation. The cosets are enumerated by $\underline{a} \in Z/\underline{r}$. If $\underline{a} \in Z/\underline{r}$ is contained in the $k-$dimensional hyperplane associated with the PE in step (1), then output coset $\underline{a}$ is also contained in the PE. Cosets of output are redundant exactly the same way that points are redundant in the computation of a single $F_{\underline{r}}$. When it is desirable, the redundancy can be eliminated in an analogous manner.

4.2 *Conclusion*

This chapter presented a new algorithm for the multiprocessor computation of the MD DFT. The algorithm is based on the paralellization of the hyperplane algorithm introduced in chapter 2. It is intended for multiprocessors connected to a single I/O channel. The target processor must support the communication functions broadcast and report.

Implementations of the algorithm are considered for two basic situations. The first is when the degree of parallelism of the target processor matches one of $M_k(\underline{n})$, $k = 1, \ldots, d-1$. Where $M_k(\underline{n})$ is the number of $k-$dimensional hyperplanes required to cover $Z/\underline{n}$. For these cases the algorithm is shown to require the time to compute one $k-$dimensional DFT plus the time to input and output the data. Computationally, the speedup of the algorithm is the ratio of a $d-$dimensional DFT to a single $k-$dimensional DFT.

The second case of the algorithm applies when the size of the machine does not match $M_k(\underline{n})$. That is, the degree of parallelism of the algorithm is not compatible with the size of the machine. For this case the hyperplane algorithm is married to the multidimensional Cooley-Tukey algorithm (MD CT). The role of the MD CT algorithm in the hybrid algorithm is to factor $F_{\underline{n}}$ into stages of $F_{\underline{s}}$ and $F_{\underline{r}}$, where each n_i is the composite $n_i = r_i s_i$. For this case the degree of parallelism is one of $M_k(\underline{r})$, for $k = 1, \ldots, d-1$. In this manner the degree of parallelism of the computation can be modified to match the size of the target processor. Like the direct method, the hybrid algorithm requires no interprocessor communication.

Chapter 5

Program Results

This research program developed a new algorithm for the multiprocessor computation of $d-$dimensional discrete Fourier transform. The main features of the algorithm are that it requires no interprocessor communication and that it is highly scalable. The motivation for this research program, and the elements that were required to realize it are described below.

This research program has been driven by the need for new algorithmic methods that can exploit the power of VLSI based multiprocessors. These machines have large computational power but limited communication bandwidth. They strongly favor algorithms that minimize interprocessor communication. This principle of locality dominates algorithm design at all levels.

Multidimensional Cooley-Tukey algorithms, and their variants, require interprocessor communication because they partition the data set at every stage of the computation. At a minimum this necessitates an interprocessor communication requirement where every processing element must exchange data with every other processing element to complete the calculation.

This research approaches the problem of parallel MD DFT computation from a new perspective. It applies a reduction rather than a partitioning algorithm to MD DFT computation. The result is that no interprocessor communication is required from input to output. This eliminates the need for an interconnection network for parallel MD DFT computation. These networks are the slowest, most costly, complex and energy inefficient elements of the multiprocessor system. The method developed in this program is nearly linear in speedup, it is highly scalable, and it requires no interprocessor communication.

The method is based on a hyperplane algorithm for MD DFT computation. The hyperplane algorithm is introduced in chapter 2. It is derived by restricting the $d-$dimensional DFT to $k-$dimensional hyperplanes of the output array. The algorithm is developed in two parts. The first part is the specification of a minimal set of $k-$dimensional hyperplanes that cover a $d-$dimensional array. These are given by theorems 2.2.1 and 2.3.1 in sections 2.2 and 2.3 respectively. The second part of the derivation is the restriction of the $d-$dimensional DFT to the hyperplanes of a covering set. The formulation of the resulting algorithm is given by equations (2.20) and (2.21) or (2.34)

and (2.35). An appendix to chapter 2 is provided that enumerates 3D and 4D cases of the algorithm.

A broadcast mode multiprocessor algorithm based on a direct mapping of the hyperplane algorithm is given in chapter 3. The main advantage of this algorithm is that it eliminates the need for interprocessor communication in MD DFT computation. The cost of eliminating the communication requirement typical of parallel MD DFT algorithms is that one addition must be performed at each processor for every input loaded into the machine. The algorithm requires a machine whose degree of parallelism equals the number of $k-$dimensional hyperplanes required to cover a $d-$dimensional array. That number is given by theorems 2.2.1 and 2.3.1 of chapter 2. This imposes limitations on the degree of parallelism of the target machine which are circumvented by the alternative mapping of chapter 4. The structure of the direct mapping algorithm is given by the procedure of figure 3.2.

Chapter 4 presents an alternative multiprocessor mapping of the hyperplane algorithm that requires no interprocessor communication and is also highly scalable. The algorithm is derived by applying hyperplane and multidimensional Cooley-Tukey methods together. The resulting algorithm uses Cooley-Tukey methods to lower the order of the MD DFT stages that the hyperplane algorithm is applied to. The degree of parallelism of the algorithm is equal to the number of hyperplanes required to cover one of the lower order MD DFT of the Cooley-Tukey factorization. Like the direct mapping algorithm the hybrid algorithm requires no interprocessor communication. The cost of eliminating interprocessor communication by this method is that one addition must be performed at each processing element for each word loaded into the machine.

Bibliography

[1] M. An, I. Gertner, M. Rofheart, and R. Tolimieri. *Discrete Fast Fourier Transforms, A Tutorial.* Advances in Applied Physics, To Appear July 1991.

[2] *AT&T DSP Parallel Processor BT-100.* AT&T, Whippany, NJ, 1988.

[3] L. Auslander, E. Feig, and S. Winograd. *New algorithms for the multidimensional discrete Fourier Transform.* IEEE Transactions Acoustics, Speech, Signal Processing, ASSP-31(2), April 1983, pp. 388–403.

[4] D. H. Bailey. *FFTs in external and hierarchical memory.* The Journal of Supercomputing. Vol. 4, 1990, pp. 23–25.

[5] K. E. Batcher. *The flip network in Staran.* Int. Conf. Par. Proc., Aug. 1976, pp. 65–71.

[6] G. D. Bergland. *A parallel implementation of the fast Fourier transform algorithm.* IEEE Transactions on Computers, C-21(4), April 1972, pp. 366-370.

[7] R. E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, Reading, Mass., 1987.

[8] N. U. Chowdary and W. Steenart. *A high speed two dimensional FFT processor.* ICASSP, San Diego Ca, 1984, pp. 4.11.1–4.11.4.

[9] J. W. Cooley and J. W. Tukey. *An algorithm for the machine calculation of complex Fourier series.* Math. Comput., Vol. 19, Apr 1965, pp. 297–301.

[10] T. Feng. *Data manipulating functions in parallel processors and their implementations.* IEEE Transactions Computers, C-23, Mar 1974, pp. 309–318.

[11] Izidor Gertner. *A new efficient algorithm to compute the two-dimensional discrete Fourier transform.* IEEE Transactions Acoustics, Speech, Signal Processing, ASSP-36(7), July 1988, pp. 1036–1050.

[12] I. Gertner and R. Tolimieri. *Fast algorithms to compute multidimensional discrete Fourier transform.* SPIE Real-Time Signal Processing Proceedings, San Diego Ca., August 1989, pp. 132–146.

[13] A. L. Gorin, L. Auslander, and A. Silberger. *Balanced computation of 2D transforms on a tree machine.* To Appear in Applied Mathematics Letters.

[14] D. B. Harris, J. H. McClellan, D. S. Chan, and H. W. Schuessler. *Vector radix fast Fourier transform.* IEEE ICASSP, Hartford, Conn. May 1977, pp. 548–551.

[15] R. W. Hockney and C. R. Jesshope. *Parallel Computers.* IOP Publishing, Bristol, England. 1989.

[16] Wayne Scott Hornick. *The Mesh of Trees Architecture for Parallel Computation.* PhD thesis, University of Illinois at Urbana-Champaign, January 1989.

[17] E. A. Hoyer, W. R. Berry. *An algorithm for the two dimensional Fourier transform.* IEEE ICASSP, Hartford, Conn. May 1977, pp. 552–555.

[18] L. H. Jamieson, P. T. Mueller, and H. J. Siegel. *FFT algorithms for SIMD processing.* J. Par. & Dist. Comp., Vol. 3, 1986, pp. 48–71.

[19] C. R. Jesshope. *The implementation of fast radix 2 transforms on array processors.* IEEE Transactions on Computers, C-29(1), Jan 1980, pp. 20–27.

[20] J. J. Kovaly. *Synthetic Aperture Radar.* Artech House, Reading MA, 1976.

[21] D. H. Lawrie. *Access and alignment of data in an array processor.* IEEE Transactions on Computers, C-24, Dec 1975, pp. 1145–1155.

[22] R. M. Mersereau, T. C. Speake. *A unified treatment of Cooley-Tukey algorithms for the evaluation of the multidimensional DFT.* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-29(5), Oct. 1981, pp. 1011–1018.

[23] V. Milutinovic, A. B. Fortes, and L. H. Jamieson. *A multiprocessor architecture for real-time computation of a class of DFT algorithms.* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-34, Oct. 1986,pp. 1301–1309.

[24] A. Norton and A. Silberger. *Parallelization and performance analysis of the Cooley-Tukey FFT algorithm on shared-memory architectures.* IEEE Transactions on Computers, C-36(5), May 1987, pp. 581–591.

[25] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, New York, 1982 (Second Edition).

[26] H. Nussbaumer, P. Quandalle. *Fast computation of discrete Fourier transforms using polynomial transforms.* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-27, April 1979, pp. 169–181.

[27] G. E. Rivard. *Direct fast Fourier transform of bivariate functions,* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-25, 1977, pp. 300-309.

[28] M. Rofheart. *Algorithms and Methods for Multidimensional Digital Signal Processing.* PhD thesis, City University of New York, March 1991.

[29] M. Rofheart and I. Gertner. *A parallel algorithm for 2D DFT computation with no interprocessor communication.* IEEE Transactions on Parallel & Distributed Systems, Vol. 1(3), July 1990, pp. 377–382.

[30] H. J. Seigel and R. J. McMillen. *The multistage cube: a versatile interconnection network.* Computer, Vol. 14, Dec. 1981, pp. 65–76.

[31] H. J. Seigel and R. J. McMillen. *Using the augmented data manipulator network in PASM.* Computer, Vol. 14, Feb. 1981, pp. 25–33.

[32] C. L. Seitz. *Concurrent VLSI architectures.* IEEE Transactions on Computers, Vol. C-33(12), Dec. 1984, pp. 1247–1264.

[33] Richard Tolimieri, Myoung An, and Chao Lu. *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, New York, 1989.

[34] Dwen-Ren Tsai and Michael Vulis. *Computing discrete Fourier transform on a rectangular data array.* IEEE Transactions Acoustics, Speech, Signal Processing. Vol. ASSP-38(2), Feb. 1990, pp. 271–276.

[35] Michael Vulis. *The weighted redundancy transform.* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-37(11), Nov. 1989, pp. 1687–1692.

[36] D. S. Wise. *Compact layouts of Banyan/FFT networks.* VLSI Systems and Computations, Computer Science Press, Rockville Md., 1981. pp. 186–195.

[37] E. L. Zapata, F. F. Rivera, I. Benavides, J. M. Carazo, R. Peskin. *Multidimensional fast Fourier transform into SIMD hypercubes.* Proc. IEE, Vol. 137(4), July, 1990, pp. 253–260.